



**Università  
di Genova**

**DIBRIS** DIPARTIMENTO  
DI INFORMATICA, BIOINGEGNERIA,  
ROBOTICA E INGEGNERIA DEI SISTEMI

# Evaluating Resilience and Trust Dynamics in Decentralized Reputation Systems

Cristian Casagrande

Master's Thesis

Università di Genova, DIBRIS  
Via Dodecaneso, 35 16146 Genova, Italy  
<https://www.dibris.unige.it/>



**Computer Science MSc**  
Software Security and Engineering Curriculum

# **Evaluating Resilience and Trust Dynamics in Decentralized Reputation Systems**

Cristian Casagrande

Advisor: Matteo Dell'Amico

Examiner: Marina Ribaudo

September, 2026

# Abstract

Platforms based on the shared and second-hand economy are becoming increasingly popular in everyday life. A common limitation of these platforms is that they rely on proprietary, siloed, and centralized reputation systems, which prevent mutual collaboration between different services. This creates significant drawbacks: a malicious user can cause issues on one platform while maintaining a clean reputation on all others, while honest users continuously face the cold-start problem when joining a new service. Dell’Amico [Del20] proposed an alternative solution based on the Web of Trust concept of Zimmermann [Zim95] to design a decentralized reputation system, allowing users to maintain a single, shared reputation across different platforms. Building upon Dell’Amico’s model, Romano [Rom25] developed a framework for trust metric computation in such decentralized networks. The goal of this master’s thesis is to evaluate the security of Romano’s framework. To achieve this purpose, we implemented a simulator that interacts directly with Romano’s work, incorporating various core components such as node behavior management, punishment mechanisms, adversarial attacks, and graphical interfaces for data analysis.

# Acknowledgments

I would like to express my sincere gratitude to my advisor, Prof. Matteo Dell'Amico, for his invaluable support, guidance, and constant availability throughout these long months of work. I am equally grateful to Prof. Marina Ribaudo for serving as my examiner and for her insightful evaluation of this thesis. My appreciation also extends to all the professors who guided me over the years, instilling in me a true passion for this discipline.

I am truly grateful to my university companions, Francesco, Henry, and Kalpa, for making these two years of study as lighthearted and enjoyable as possible. To my closest friends, who are an essential part of my everyday life: Alessio, Elena, Erich, Fù, Isaac, Jiawei, Luca, Matteo, and Sally, thank you for constantly putting up with my nonsense. It is largely because of you that I am the person I am today.

Words cannot fully express my gratitude to my mother and my sister for everything they have done for me, for their endless sacrifices, and for always believing in me. My heartfelt thanks also go to the rest of my family for their unwavering love and support. Most importantly, I dedicate this milestone to my girlfriend, Giorgia, my grandfather, Umberto, and my uncle, Mingo. Everything I have accomplished is above all for you. I love you.

# Declaration of AI Usage

In compliance with the University of Genoa guidelines on the use of Artificial Intelligence (September 2025), the author declares that generative AI tools were used solely for language editing, text refining, and improving narrative fluency. All underlying concepts, research questions, simulation designs and data interpretations are entirely the original work of the author.

# Table of Contents

|                  |                                              |           |
|------------------|----------------------------------------------|-----------|
| <b>Chapter 1</b> | <b>Introduction</b>                          | <b>9</b>  |
| <b>Chapter 2</b> | <b>Background &amp; Related Work</b>         | <b>11</b> |
| 2.1              | Decentralized Systems . . . . .              | 11        |
| 2.2              | Trust and Reputation . . . . .               | 11        |
| 2.2.1            | Trust Transitivity . . . . .                 | 12        |
| 2.2.2            | Web of Trust . . . . .                       | 13        |
| 2.3              | Routing Metrics . . . . .                    | 14        |
| 2.3.1            | Graph-Based Metrics . . . . .                | 14        |
| 2.3.2            | Embedding-Space Metrics . . . . .            | 16        |
| 2.4              | Attacks on Trust . . . . .                   | 19        |
| 2.4.1            | Sybil Attacks . . . . .                      | 20        |
| 2.4.2            | Self-Promoting Attacks . . . . .             | 21        |
| 2.4.3            | Whitewashing Attacks . . . . .               | 23        |
| 2.4.4            | Slander Attacks . . . . .                    | 24        |
| 2.4.5            | Betrayal Attacks . . . . .                   | 25        |
| 2.4.6            | Orchestrated Attacks . . . . .               | 26        |
| 2.5              | Discrete-Event Simulation . . . . .          | 27        |
| 2.5.1            | Core System Abstractions . . . . .           | 28        |
| 2.5.2            | Event-Oriented Execution Mechanism . . . . . | 28        |

|                  |                                                                 |           |
|------------------|-----------------------------------------------------------------|-----------|
| 2.5.3            | Event Scheduling and Simulation Loop . . . . .                  | 28        |
| 2.5.4            | Stochastic Modeling and Event Distributions . . . . .           | 29        |
| 2.5.5            | Simulation Control and Data Collection . . . . .                | 31        |
| <b>Chapter 3</b> | <b>Methodology</b>                                              | <b>32</b> |
| 3.1              | Simulated System and Operational Model . . . . .                | 33        |
| 3.1.1            | Node Behavioral Profiles and Adversarial Model . . . . .        | 33        |
| 3.1.2            | The Trust Threshold and Graph State Separation . . . . .        | 34        |
| 3.1.3            | Transaction Execution and Node Connectivity Rules . . . . .     | 34        |
| 3.2              | Routing Algorithms Implementation . . . . .                     | 35        |
| 3.2.1            | Standard Graph-Based Metrics . . . . .                          | 36        |
| 3.2.2            | Embedding-Space Metrics . . . . .                               | 37        |
| 3.3              | Formalization of Adversarial Attacks . . . . .                  | 39        |
| 3.3.1            | Strategic Betrayal Attack . . . . .                             | 39        |
| 3.3.2            | Collusion Attack . . . . .                                      | 40        |
| 3.3.3            | Slander Attack . . . . .                                        | 41        |
| 3.4              | Countermeasures to Malicious Behavior . . . . .                 | 42        |
| 3.4.1            | Mitigation of Slander Attacks via Localized Consensus . . . . . | 42        |
| 3.4.2            | Single Punishment . . . . .                                     | 43        |
| 3.4.3            | Path Punishment . . . . .                                       | 44        |
| 3.4.4            | Recommender Punishment . . . . .                                | 45        |
| 3.5              | Technical Project . . . . .                                     | 46        |
| 3.5.1            | Libraries . . . . .                                             | 46        |
| 3.5.2            | Custom Performance Optimization . . . . .                       | 47        |
| 3.5.3            | Discrete Event Simulation Engine and System Events . . . . .    | 47        |
| 3.5.4            | Experiment Configuration . . . . .                              | 49        |
| 3.5.5            | Data Processor . . . . .                                        | 51        |
| 3.6              | Modularity and Extensibility . . . . .                          | 52        |

|                  |                                                              |            |
|------------------|--------------------------------------------------------------|------------|
| <b>Chapter 4</b> | <b>Results</b>                                               | <b>54</b>  |
| 4.1              | Experimental Overview . . . . .                              | 54         |
| 4.2              | Parameter Configuration . . . . .                            | 54         |
| 4.2.1            | Static Parameters . . . . .                                  | 55         |
| 4.2.2            | Dynamic Parameters . . . . .                                 | 56         |
| 4.3              | System Behavior Under Attack and Countermeasures . . . . .   | 57         |
| 4.3.1            | Normal Run Scenario . . . . .                                | 59         |
| 4.3.2            | Slander Attack Scenario . . . . .                            | 61         |
| 4.3.3            | Collusion Attack Scenario . . . . .                          | 63         |
| 4.3.4            | Slander-Collusion Attack Scenario . . . . .                  | 65         |
| 4.3.5            | Trade Pooling Defense . . . . .                              | 67         |
| 4.3.6            | Single punishment Defense . . . . .                          | 75         |
| 4.3.7            | Recommender Punishment Defense . . . . .                     | 86         |
| 4.3.8            | Path Punishment Defense . . . . .                            | 91         |
| 4.3.9            | Transaction Analysis of Honest and Malicious Nodes . . . . . | 91         |
| <b>Chapter 5</b> | <b>Conclusion</b>                                            | <b>103</b> |

# Chapter 1

## Introduction

Nowadays, platforms dedicated to the sharing and second-hand economy, such as Vinted, BlaBlaCar, and Airbnb, depend on direct peer-to-peer interactions between users who do not know each other. In these environments, deploying a reliable reputation system is essential to establish **trust** and **encourage cooperation**. Typically, these systems use feedback mechanisms to build a public history of user reliability before a transaction takes place.

Despite their effectiveness, most existing platforms rely on **centralized** reputation management, locking user data within proprietary **silos**. This model introduces structural limitations: honest users face a **cold-start** problem when joining a new service, as they cannot transfer their reputation history. Conversely, malicious actors can exploit one platform while preserving a clean reputation on others.

To solve this problem, Dell’Amico [Del20] proposed a **decentralized** architecture based on Zimmermann’s **Web of Trust (WoT)** concept [Zim95]. By modeling the network as a graph  $G = (V, E)$ , where  $V$  represents users and  $E$  pairwise trust relationships derived from past interactions, this approach enables cross-platform reputation portability. In this graph, trust is formulated as a personalized scoring function  $r : V \times V \rightarrow \mathbb{R}$ , allowing an evaluating node  $a$  to compute the trustworthiness of a target node  $b$  from its local perspective.

To make this model viable in large-scale networks, Romano [Rom25] developed a framework to compute **trust metrics** in P2P networks without requiring global graph visibility. By combining graph embeddings with localized search algorithms, Romano showed that nodes can estimate personalized trust scores using only local neighbor information.

However, Romano’s framework **assumed that all network participants behave honestly**. In open P2P environments, reputation systems are vulnerable to malicious manipu-

lation. Without a central authority to audit transactions, malicious nodes can manipulate feedback, launch coordinated attacks such as **Slander** or **Collusion**, and corrupt trust metrics.

Taking this context into account, the main goal of this thesis is to evaluate the **security** and **resilience** of Romano’s framework under adversarial scenarios. To achieve this, we developed an **event-driven simulation environment** extending Romano’s architecture that models dynamic node behaviors, executes representative P2P attacks, implements topological punishment mechanisms, and gathers performance metrics for analysis.

The remainder of this thesis is structured as follows. Chapter 2 establishes the theoretical foundation by examining discrete-event simulation models, attack vectors in distributed networks, and defensive countermeasures for reputation systems. Building on these concepts, Chapter 3 presents our methodology, describing the overall architecture of the simulation framework, its internal components, configuration parameters, and practical implementation. Chapter 4 evaluates the system across various experimental setups, providing an in-depth analysis of the collected security and performance metrics. The thesis concludes in Chapter 5 with a summary of our main findings and potential avenues for future research.

# Chapter 2

## Background & Related Work

This chapter establishes the theoretical foundations for this work. It begins with an overview of **decentralized P2P architectures** (Section 2.1), highlighting why distributed environments require robust mechanisms for **trust, reputation, and transitivity** (Section 2.2). To understand how these mechanisms perform in hostile settings, the discussion turns to primary **attack vectors** and their mitigation strategies (Section 2.4). Finally, it introduces **Discrete-Event Simulation (DES)** (Section 2.5) as the methodological framework used to model and evaluate these dynamic systems.

### 2.1 Decentralized Systems

A decentralized system consists of a network of autonomous nodes that interact directly without relying on a central authority. Such systems typically adopt a **pure Peer-to-Peer (P2P) architecture**, where participants can dynamically join, disconnect, or leave the network without disrupting service continuity [Sch01]. Within this framework, individual nodes possess only **local visibility** of the overall network topology. Consequently, in the absence of a trusted third party to validate identities or guarantee exchanges, nodes must assess peer credibility independently, rendering **decentralized trust and reputation mechanisms** essential for secure peer interactions.

### 2.2 Trust and Reputation

To model interactions in a decentralized network, it is essential to first clarify what "trust" actually means in practice. As described in the paper by Jøsang et al. [JIB07], trust

is generally divided into two core operational concepts: **reliability trust** and **decision trust**.

Reliability trust focuses purely on expected behavior. As defined by Gambetta [Gam90], it represents the subjective probability that a node expects a peer to perform a specific action correctly. In simple terms, it answers the question: *“How likely is this peer to behave as expected?”*

However, high reliability alone is not always enough to trigger an interaction. This is where decision trust comes in [MC96]. Decision trust represents a node’s actual choice to depend on another peer while accepting the situational risk. The key distinction lies in the potential impact of a failure: even if a peer has a 99% reliability score, a node might still decline to interact if a single failure could cause a critical loss. Decision trust, therefore, balances estimated reliability with risk tolerance and potential consequences.

Beyond this classification, a fundamental distinction exists between private **trust** and public **reputation**:

- **Trust** is private and subjective, built primarily on direct past experience. Direct experience always carries the highest weight; a node will trust a peer it knows to be honest regardless of what the rest of the network thinks.
- **Reputation** is public and collective, aggregated from referrals across the network. Its primary role is bootstrapping: when two nodes have no prior interaction history, public reputation provides the initial baseline needed to decide whether to connect.

### 2.2.1 Trust Transitivity

In decentralized networks, an individual node rarely possesses direct interaction history with every potential counterpart. To overcome this limitation and enable nodes to evaluate unfamiliar entities, systems rely on **trust transitivity** [JIB07; Del20]. This principle allows an evaluating node  $A$  to establish indirect trust in an unknown node  $C$  by leveraging its existing relationship with a trusted intermediate peer  $B$ .

Formally, this propagation pattern comprises three core relations:

- **Direct trust:** Node  $A$  holds direct trust in node  $B$  ( $A \rightarrow B$ ).
- **Direct trust and referral:** Node  $B$  holds direct trust in node  $C$  and refers it to node  $A$  ( $B \rightarrow C$ ).
- **Derived trust:** Node  $A$  derives indirect trust in node  $C$  through node  $B$  ( $A \dashrightarrow C$ ).

Combining  $A$ 's trust in  $B$  with  $B$ 's assessment of  $C$  to formulate a new evaluation is known as **transitive derivation**. However, trusting a peer to execute a service does not automatically imply trusting their judgment in recommending others. For this derivation to remain valid, two fundamental semantic constraints must hold [JIB07]:

- **Referral trust:** Node  $A$  must trust node  $B$  to provide honest and accurate recommendations.
- **Reliability trust:** Node  $B$  must trust node  $C$  to execute the target action correctly.

To provide a clearer visual intuition of this mechanism, Figure 2.1 illustrates the transitive trust propagation pattern across intermediate nodes.

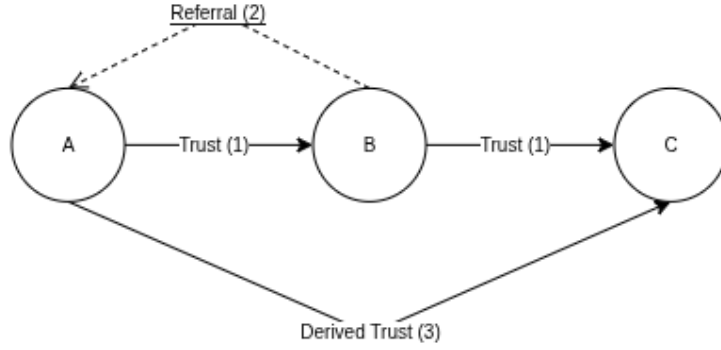


Figure 2.1: The transitive trust process evolves through three distinct stages. Initially, directed trust links exist from node  $A$  to  $B$ , as well as from  $B$  to  $C$ . Building on this foundation,  $B$  provides a referral to  $A$  regarding  $C$ , allowing  $A$  to evaluate  $C$ 's credibility and compute a derived trust score. Ultimately, this derived trust leads to a direct transaction from  $A$  to  $C$ , establishing a new directed connection.

### 2.2.2 Web of Trust

The Web of Trust (WoT) framework, originally introduced by Zimmermann [Zim95], replaces traditional **centralized Certificate Authorities** with a fully distributed model. Instead of relying on a single root entity, credibility is built organically through pairwise interactions. Each node acts as its own local authority, establishing direct trust links with peers based on verified experience. As these individual relationships overlap across the network, they form a web that enables participants to evaluate unfamiliar entities by tracing trust paths through trusted intermediaries, making trust evaluation inherently subjective and local.

This paradigm has seen practical adoption in platforms like Epinions, where users construct personalized trust networks to weight peer evaluations [JIB07].

Formally, a Web of Trust is represented as a directed graph  $G = (V, E)$ , where  $V$  denotes the set of network participants and  $E$  represents the pairwise trust relationships established between them. Within this graph, trust is expressed through a **personalized scoring function**

$$r : V \times V \rightarrow \mathbb{R}$$

which enables an evaluating node  $a \in V$  to compute a local trustworthiness score for a target node  $b \in V$  purely from its own perspective.

## 2.3 Routing Metrics

This section introduces the routing metrics used to evaluate node reliability and trust within a **Web of Trust**, building upon the principle of **trust transitivity**. To evaluate peers without direct interaction history, pathfinding algorithms traverse the graph to discover transitive trust paths and compute aggregate scores.

These routing approaches generally fall into two main categories: **graph-based** metrics, which operate directly on the network topology, and **embedding-space** metrics. Embedding methods overcome the limited local visibility of decentralized peers by mapping nodes into target representation spaces, such as geometric coordinates or tree-based structures. This allows trust distances to be calculated locally without requiring global topological knowledge. For the embedding approaches, this work focuses specifically on the metrics evaluated in the work of Romano [Rom25].

### 2.3.1 Graph-Based Metrics

Graph-based metrics evaluate peer reliability by operating on the global topology of the Web of Trust, modeled as a weighted directed graph  $G = (V, E)$ . To derive indirect trust between unconnected nodes, these metrics analyze structural properties along intermediate paths. Depending on the strategy, evaluation relies on **path distance**, **network flow capacity**, or **stochastic trust propagation** across the graph.

#### Shortest Path Distance

The Shortest Path Distance metric models edge weights as **routing costs**, where lower values indicate higher direct trust. Let  $G = (V, E)$  be the network graph, where  $w : E \rightarrow$

$\mathbb{R}_{\geq 0}$  is a weight function assigning a non-negative routing cost  $w(u, v)$  to each directed edge  $(u, v) \in E$ .

Let  $\mathcal{P}_{s,d}$  denote the set of all directed paths  $p = (v_0, v_1, \dots, v_k)$  from a source node  $s = v_0$  to a target node  $d = v_k$ . Using Dijkstra's algorithm [Dij59], the optimal routing path  $p^*$  minimizes the cumulative cost:

$$p^* = \arg \min_{p \in \mathcal{P}_{s,d}} \sum_{(u,v) \in p} w(u, v) \quad (2.1)$$

The cumulative path distance  $C^*$  is defined as:

$$C^* = \begin{cases} \sum_{(u,v) \in p^*} w(u, v) & \text{if } \mathcal{P}_{s,d} \neq \emptyset, \\ \infty & \text{if } \mathcal{P}_{s,d} = \emptyset. \end{cases} \quad (2.2)$$

**Value Interpretation in a Web of Trust:** Within a Web of Trust, the cumulative cost  $C^*$  penalizes both **poor reputation scores** and **long recommendation chains**. Because each intermediate hop introduces potential uncertainty or trust degradation, an optimal path minimizes individual link costs while restricting total length, preserving **trust** across the recommender chain.

## Maximum Flow

The Maximum Flow metric models edge weights as **network capacities**, where higher values indicate stronger direct trust or greater historical interaction volume. Let  $G = (V, E)$  be the network graph, where  $c : E \rightarrow \mathbb{R}_{\geq 0}$  is a capacity function assigning a non-negative capacity  $c(u, v)$  to each directed edge  $(u, v) \in E$ .

For a source node  $s \in V$  and a target node  $d \in V$ , a valid network flow is a function  $f : E \rightarrow \mathbb{R}_{\geq 0}$  satisfying the capacity constraint:

$$0 \leq f(u, v) \leq c(u, v), \quad \forall (u, v) \in E \quad (2.3)$$

and the flow conservation constraint at every intermediate node:

$$\sum_{v \in V} f(u, v) = \sum_{v \in V} f(v, u), \quad \forall u \in V \setminus \{s, d\} \quad (2.4)$$

The objective is to determine the maximum flow value  $F^*$ , defined as the maximum net flow that can be routed from  $s$  to  $d$ :

$$F^* = \max_f \sum_{v \in V} f(s, v) \quad (2.5)$$

Based on the maximum flow formulation by Ford and Fulkerson [FF56],  $F^*$  is computed using the Edmonds-Karp algorithm [EK72], which iteratively identifies shortest augmenting paths from  $s$  to  $d$  in the residual graph via Breadth-First Search (BFS) until the network reaches saturation.

**Value Interpretation in a Web of Trust:** Within a Web of Trust, the total flow  $F^*$  quantifies the **structural redundancy** and **bottleneck capacity** of trust between  $s$  and  $d$ . Unlike single-path metrics, a higher maximum flow indicates that trust propagates through multiple **independent, high-capacity parallel paths**. Consequently, a larger  $F^*$  guarantees that the overall recommendation relationship does not depend on a single critical intermediary, mitigating the risk of a **single point of failure**.

### Personalized PageRank

The Personalized PageRank metric, based on the original PageRank algorithm introduced by Brin and Page [BP98] and adapted for personalized evaluation by Haveliwala [Hav02], models trust propagation as **random walks** over the network graph  $G = (V, E)$  to measure the relative reputation of candidate nodes from the perspective of a specific source node  $s$ .

Let  $\mathbf{PR}$  represent the stationary probability distribution vector, and let  $\mathbf{W}$  be the row-normalized transition probability matrix, where edge transition probabilities are weighted proportionally to direct reputation scores. Given a damping factor  $\alpha = 0.85$  and a personalization vector  $\mathbf{v}$ , the PageRank vector satisfies the linear system:

$$\mathbf{PR} = \alpha \mathbf{PRW} + (1 - \alpha) \mathbf{v} \quad (2.6)$$

The personalization vector  $\mathbf{v}$  concentrates the teleportation probability entirely on the source node  $s$ , where  $v_u = 1$  if  $u = s$  and  $v_u = 0$  otherwise. Unlike path-bound metrics, PageRank scores are computed for all active nodes simultaneously, from which the source node extracts the target’s specific score  $PR_d$ .

**Value Interpretation in a Web of Trust:** Within a Web of Trust, the score  $PR_d$  measures the **localized centrality** and **structural proximity** of the target node  $d$  relative to the source  $s$ . A higher PageRank score indicates that a random walk starting from the source has a high probability of reaching the target via trusted links. This demonstrates that the target is well-integrated into the source’s local network and validated by a cluster of active nodes, reducing the risk of **localized manipulation** or **collusion attacks**.

### 2.3.2 Embedding-Space Metrics

Embedding-space metrics evaluate peer reliability by mapping network nodes into **virtual coordinate layouts**, addressing the limited local visibility of decentralized peers. To

derive indirect trust between unconnected nodes, pathfinding strategies navigate these representation spaces by evaluating **tree or geometric distances** toward the target node. When a strategy isolates a simple path  $p$  without computing a localized graph metric, the underlying trust score corresponds directly to the **path length**  $|p|$ , following the Shortest Path Distance principles from Section 2.3.1.

### 2.3.2.1 Tree-Based Embedding Metrics

Under the **Spanning Trees Graph Embedding (STGE)** model [RBS16a; RBS16b], the network topology is mapped onto a set of rooted spanning trees  $\mathcal{T}$ . Pathfinding routines rely on **structural tree distances** to select the closest layout between the source  $s$  and destination  $d$ , routing transactions locally without requiring global knowledge of the graph  $G = (V, E)$ .

#### Tree Best-First Search

The Tree Best-First Search adapts the classical Beam Search paradigm to prioritize network exploration using structural tree proximity [RBS16a]. Let  $T_1 \in \mathcal{T}$  define the selected rooted spanning tree, where the distance function  $d_{T_1}(u, v)$  quantifies the path length separating two nodes within the tree structure. The algorithm explores the local neighborhood by maintaining a candidate queue bounded by a maximum beam width  $W$ . Intermediate hops are iteratively selected to minimize the tree distance to the target node  $d$ :

$$v_{\text{next}} = \arg \min_{v \in \text{Queue}} d_{T_1}(v, d) \quad (2.7)$$

Upon reaching the destination node, the search terminates and returns the identified path  $p$ . If no valid candidates are retrieved, local backtracking is executed.

#### Tree Greedy Depth-First Search

Tree Greedy Depth-First Search combines classical greedy routing with depth-first backtracking [Cla+01]. Guided by structural tree distances, the algorithm executes a localized depth-first exploration directly over the graph  $G = (V, E)$ . From the current node  $u$ , the search evaluates immediate neighbors  $\mathcal{N}(u)$  and greedily selects the peer that minimizes the tree distance to  $d$ :

$$v_{\text{next}} = \arg \min_{v \in \mathcal{N}(u)} d_{T_1}(v, d) \quad (2.8)$$

The exploration proceeds recursively along the most promising branch without maintaining a global priority queue. If a dead end is encountered, local backtracking is performed until a valid route  $p$  to  $d$  is isolated and returned.

### Tree Ping-Pong Depth-First Search

Tree Ping-Pong Depth-First Search [RBS16b] performs tree-guided exploration to discover multiple paths between the source  $s$  and destination  $d$ . To guide pathfinding, the algorithm selects the rooted spanning tree layout  $T_1 \in \mathcal{T}$  that minimizes the tree distance  $d_{T_1}(s, d)$  to the target node. Instead of stopping once a single path is found, the search continues exploring nearby nodes up to a predefined limit to uncover alternative routes. This process merges all discovered paths into a **localized subgraph**  $G_{\text{explored}} \subseteq G$ . A localized trust metric  $M(d)$  is then calculated over this bounded subgraph:

$$M(d) = \begin{cases} \text{MaxFlow}(G_{\text{explored}}, s, d), & \text{for Maximum Flow evaluation [CF05],} \\ \text{PPR}(G_{\text{explored}}, s)_d, & \text{for Personalized PageRank evaluation [DC10].} \end{cases} \quad (2.9)$$

The search concludes by returning the computed trust score  $M(d)$ .

**Value Interpretation in an Embedded Web of Trust:** Expanding exploration up to a limit ensures  $G_{\text{explored}}$  captures local path diversity. Within this isolated subgraph, Maximum Flow confirms **structural path redundancy**, while Personalized PageRank measures the destination’s **structural proximity** to the source’s trust domain without requiring global graph queries.

#### 2.3.2.2 Continuous Space Embedding Metrics

Under the **Social Coordinate System (SoCS)** approach [KLT11], node positions map to a continuous geometric space within the layout. Pathfinding routines rely on **spatial distance metrics** to route transactions locally without requiring global knowledge of the graph  $G = (V, E)$ .

#### Best-First Search with NoNRS Optimization

The Best-First Search strategy with **NoNRS** optimization [KLT11] utilizes continuous spatial coordinates combined with a density correction to navigate past sparse regions in the embedding layout. Let  $\mathbf{x}_u$  denote the coordinate vector of node  $u$ , evaluated via the Euclidean distance  $\|\mathbf{x}_u - \mathbf{x}_v\|$ . To bypass empty topological zones, intermediate hops are selected from the queue to minimize a density-corrected spatial distance to  $d$ :

$$v_{\text{next}} = \arg \min_{v \in \text{Queue}} f_{\text{NoNRS}}(\|\mathbf{x}_v - \mathbf{x}_d\|) \quad (2.10)$$

Upon discovering a route to the destination node, the algorithm returns the path  $p$ .

### Greedy Depth-First Search with SPS

Greedy Depth-First Search with Shortest Path on Subgraph (SPS) [KLT11] performs continuous geometric exploration over  $G = (V, E)$ , selecting neighboring nodes that minimize the spatial distance to  $d$ . Once the destination is reached, the algorithm identifies the shortest path strictly within the sub-network  $G_{\text{explored}}$  uncovered during the greedy search phase:

$$p_{\text{SPS}} = \arg \min_{p \subseteq G_{\text{explored}}} |p| \quad (2.11)$$

The routine returns the optimized path  $p$ .

### Continuous Geometric Ping-Pong Depth-First Search

Continuous Geometric Ping-Pong Depth-First Search [KLT11] performs spatial geometric exploration to discover multiple paths between the source  $s$  and destination  $d$ . To guide pathfinding, the algorithm evaluates spatial distances within the continuous coordinate layout toward the target node. Instead of stopping once a single path is found, the search continues exploring nearby nodes up to a predefined limit to uncover alternative routes. This process merges all discovered paths into a **localized subgraph**  $G_{\text{explored}} \subseteq G$ . A localized trust metric  $M(d)$  is then calculated over this bounded subgraph:

$$M(d) = \begin{cases} \text{MaxFlow}(G_{\text{explored}}, s, d), & \text{for Maximum Flow evaluation [CF05],} \\ \text{PPR}(G_{\text{explored}}, s)_d, & \text{for Personalized PageRank evaluation [DC10].} \end{cases} \quad (2.12)$$

The search concludes by returning the computed trust score  $M(d)$ .

**Value Interpretation in an Embedded Web of Trust:** As before, expanding exploration up to a limit ensures  $G_{\text{explored}}$  captures local path diversity. Within this isolated subgraph, Maximum Flow confirms **structural path redundancy**, while Personalized PageRank measures the destination’s **structural proximity** to the source’s trust domain without requiring global graph queries.

## 2.4 Attacks on Trust

While open access and pseudonymity give P2P networks scalability and resilience, they also make them vulnerable to attacks. Without central identity verification, malicious nodes can manipulate trust metrics or consume shared resources without contributing. Securing a decentralized reputation system requires analyzing these attack vectors and designing targeted countermeasures.

### 2.4.1 Sybil Attacks

A Sybil attack occurs when a single malicious node creates and controls multiple **pseudonymous identities** in a decentralized network [Dou02]. As shown in Figure 2.2, this allows an attacker to subvert **decentralized trust** by tricking the system into treating a single physical entity as multiple independent peers.

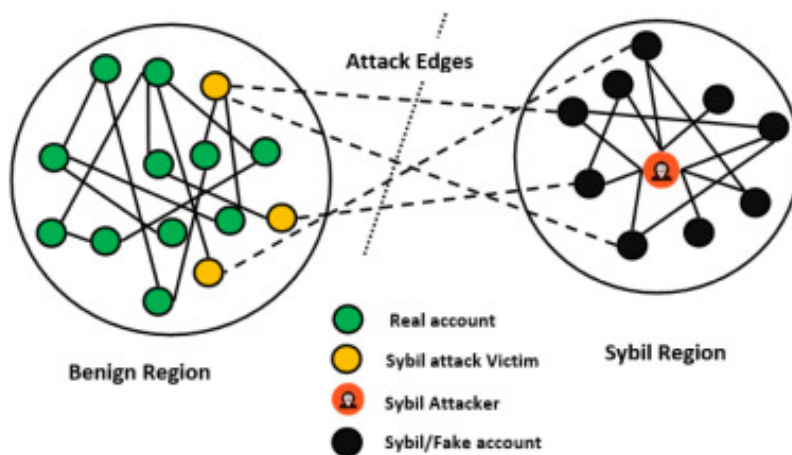


Figure 2.2: Graph representation of a Sybil attack topology. Source: Jethava and Rao [JR22].

#### Mechanism and System Impact

In open networks, creating a new identity is easy. Because nodes interact using network identifiers (**identities**) rather than physical hardware (**entities**), protocols assume that each physical entity controls only one identity. A **Sybil attack** breaks this assumption by allowing a single entity to run multiple fake identities.

By running multiple identities, an attacker pretends to be many separate nodes. This breaks features that rely on **node independence**, such as storing data across different peers or collecting independent votes.

Douceur [Dou02] showed that without a central authority, completely preventing Sybil attacks is practically impossible. It would require three unrealistic conditions: equal resource capacity across all nodes, simultaneous identity verification, and a majority of honest actors.

Since these conditions rarely hold in practice, Sybil identities usually serve as a foundation for **higher-level reputation attacks** [HZN09]:

- **Voting and Consensus Manipulation:** Tricks the system into counting multiple votes from a single physical entity, corrupting decision-making processes.
- **Amplification of Collusive Attacks:** Allows a single adversary to build fake node groups to boost its own reputation or coordinate fake negative ratings against honest peers without needing real accomplices.
- **Network Isolation:** Surrounds a targeted honest peer with malicious identities, controlling all its incoming referrals and recommendations.

## Mitigation Strategies

Defending against Sybil attacks relies on linking identity creation to scarce resources or using trust relationships [Dou02]:

- **Resource Challenges:** Nodes test remote peers with demanding tasks or financial commitments, such as Proof-of-Work puzzles, storage verification, or stake deposits [Dou02; Row+07].
- **Web of Trust Limits:** Protocols restrict recommendation depth and trust propagation to stop fake identities from spreading trust across the network graph [CF05; Yu+06; Yu+08; DM08].
- **Identity Certification:** Using trusted identity providers, hardware keys, or IP addresses ties identities to physical devices, though this reduces decentralization [Dou02].

### 2.4.2 Self-Promoting Attacks

A self-promoting attack occurs when malicious entities attempt to **artificially inflate their own reputation scores** within the network [HZN09]. While a lone attacker can attempt to alter its own ratings, self-promotion in decentralized environments is primarily executed through **collusive rings**, where a group of cooperating adversaries or a single adversary operating a cluster of Sybil identities systematically exchange false positive evaluations to subvert reputation metrics.

## Mechanism and System Impact

In its simplest form, an isolated attacker exploits unauthenticated feedback channels to fabricate positive ratings about itself. However, because modern decentralized systems often

employ cryptographic data verification, self-promotion relies predominantly on **collusive behavior** [HZN09].

In a collusive self-promoting attack, multiple controlled identities form a closed feedback loop. By generating high volumes of synthetic events and mutually issuing maximum positive ratings, colluders boost their collective reputation at a rate significantly faster than legitimate network activity. Furthermore, attackers can operate as **moles** [Fel+04]: they behave honestly toward select peers to gain initial trust, while simultaneously colluding with their ring to manipulate aggregation pipelines and shield each other from penalties.

The primary system impacts of collusive self-promoting attacks include:

- **Collusive Reputation Boosting:** By synthesizing fake transactions within a controlled ring, colluding nodes reach artificially high trust scores without providing actual service quality to honest peers.
- **Dilution and Masking of Negative Ratings:** A high volume of positive feedback generated continuously inside the collusive group dilutes past negative scores, effectively neutralizing penalization mechanisms for misbehaving nodes.
- **Subversion of Peer Selection:** Trusted node selection is corrupted across the network, deceiving honest participants into choosing high-reputation colluders over genuinely reliable peers.

## Mitigation Strategies

Defending against collusive self-promotion requires verifying transaction authenticity and identifying suspicious graph topologies [HZN09]:

- **Proof of Transaction:** Protocols enforce cryptographic proofs of completed exchanges before accepting positive feedback, preventing colluders from generating arbitrary ratings without executing real tasks [HZN09].
- **Graph-Based Clique Detection:** Algorithms analyze network interaction graphs to identify tightly coupled clusters of nodes that interact almost exclusively among themselves, reducing the weight of internal recommendations [Cor+01; HZN09].
- **Sybil Resistance Mechanisms:** Entry costs, deposit schemes, or resource-based validation limit an adversary’s capacity to instantiate the multiple identities required to build collusive rating rings [Dou02; HZN09].
- **Decoupled and Robust Aggregation:** Computation logic separates rating storage and calculation from evaluating peers, preventing colluding nodes from manipulating aggregated scores during dissemination [Fel+04; HZN09].

### 2.4.3 Whitewashing Attacks

A whitewashing attack occurs when an uncooperative or low-performing node intentionally **discards its current identity** and rejoins the network under a **fresh pseudonym** to erase accumulated negative ratings or bypass penalties [Fel+06].

#### Mechanism and System Impact

Whitewashing is made possible by **cheap pseudonyms**, where creating a new identity incurs insignificant cost. Because a returning whitewasher is computationally indistinguishable from a legitimate newcomer, an attacker can continuously reset its history. This leads to three primary impacts on the network:

- **Persistent Free-Riding:** Malicious nodes consume shared resources and reset their identity as soon as their reputation score drops, permanently evading penalties.
- **Exploitation of Bootstrapping Trust:** Fresh identities often receive default privileges or optimistic initial scores, which whitewashers repeatedly exploit.
- **Erosion of Cooperation:** Honest nodes become suspicious of all unrated participants, reducing overall peer-to-peer interactions across the network.

This forces reputation systems into a structural trade-off: imposing penalties on misbehaving nodes forces the protocol to penalize all incoming participants indiscriminately.

As analyzed by Feldman et al. [Fel+06], penalizing all newcomers introduces a **social loss** by degrading the service quality of honest new members. The impact of this social loss is strongly tied to **user turnover**: in high-turnover environments, punishing newcomers severely degrades overall system efficiency, while failing to penalize them allows uncooperative nodes to exploit the network indefinitely.

#### Mitigation Strategies

To neutralize whitewashing without harming honest participants, protocols employ three primary countermeasures:

- **Identity Acquisition Costs:** Introducing an explicit identity creation cost  $c > 0$  (e.g., computational Proof-of-Work or entry deposits) makes repeated resets economically unviable. Whitewashing is neutralized as long as the penalty differential between free-riders ( $p_{\text{free-rider}}$ ) and newcomers ( $p_{\text{newcomer}}$ ) satisfies  $p_{\text{free-rider}} - p_{\text{newcomer}} \leq c$  [Fel+06].

- **Initial Trust Constraints:** Assigning conservative default reputation scores (e.g., 0.5) or requiring initial probationary periods limits the privileges a fresh identity can immediately exploit upon entering the network [Fel+06].
- **Graph-Based Web of Trust:** In trust-graph architectures, a new identity starts with zero incoming trust edges from established participants. Resetting an identity erases past negative history, but also strips the node of any established trust paths, forcing it to rebuild credibility from scratch [HZN09].

#### 2.4.4 Slander Attacks

A slander attack, often designated as a bad-mouthing attack, occurs when one or more malicious nodes intentionally broadcast **false negative ratings** to degrade the trust scores of honest participants [JIB07; Vel+08].

##### Mechanism and System Impact

In decentralized systems, nodes leverage **peer recommendations** to evaluate participants with whom they have no prior interaction history. Slander attacks exploit this trust mechanism by spreading **corrupted evaluations** throughout the network.

The severity of a slander attack is significantly amplified when malicious entities coordinate their behavior in a **lying collusion attack** [Vel+08]. By submitting synchronized negative evaluations, colluding nodes **artificially establish a negative consensus**, tricking evaluating peers into avoiding honest participants.

The primary system impacts of slander attacks include:

- **Exclusion of Honest Participants:** Honest nodes face unfair penalties based on fake ratings, isolating them from taking part in normal network tasks and transactions.
- **Loss of Efficiency:** Lowering the reputation scores of honest peers forces evaluating nodes to rely on sub-optimal or less reputable counterparts, degrading overall network performance.
- **High Impact with Low Effort:** Unlike self-promotion attacks that try to boost an attacker’s own score, slander attacks target honest nodes directly, causing heavy damage without needing many resources.

## Mitigation Strategies

To protect decentralized reputation systems against slander and lying collusion, protocols integrate robust filtering and weighting mechanisms:

- **Recommender Trust Weighting:** Recommendations are weighted according to how much the evaluating node trusts the recommender itself. Feedback coming from unknown or low-reputation nodes is heavily reduced or ignored [Vel+08].
- **Relationship Maturity:** The impact of a recommendation depends on how long the two nodes have been connected. Long-established relationships carry more weight than newly formed links, stopping new malicious nodes from mounting effective slander attacks [Vel+08].
- **Prioritizing Direct Experience:** Direct past interactions are given much higher priority than indirect recommendations. This prevents second-hand feedback from overriding real, locally verified behavior [Vel+08].
- **Statistical Outlier Filtering:** Statistical techniques, such as tracking standard deviation  $\sigma$ , are used to detect and filter out ratings that deviate significantly from a node's historical score [JIB07].

### 2.4.5 Betrayal Attacks

A betrayal attack occurs when a **trusted node** suddenly changes its behavior and starts acting maliciously or sending false information to other peers [Fun11].

#### Mechanism and System Impact

A betrayal attacker acts honestly for a long time to build a **high reputation score**. Once it gains high trust, it uses its position to attack the system (for example, by sending bad data or false alerts) before the network notices the change [Fun11].

This attack works because standard reputation systems rely heavily on past positive behavior. As a result, a previously trusted node can cause damage while keeping its high score for a short time.

The main impacts of betrayal attacks include:

- **Misuse of High Reputation:** The attacker uses its earned trust to pass security checks and spread false data without getting blocked immediately.

- **Slow Score Drop:** Systems that do not reduce scores quickly fail to stop the attacker in time.
- **Impact on Decision-Making:** Honest nodes trust the attacker based on old scores, leading them to accept bad data or make incorrect security decisions.

## Mitigation Strategies

Defending against betrayal attacks requires strict rules where trust drops quickly upon detecting bad behavior:

- **Asymmetric Trust Updates:** Systems enforce a **slow-start, fast-drop** rule. High trust takes a long time to build, but a single bad action causes the score to drop immediately [HZN09].
- **Direct Experience Evaluation:** Nodes prioritize their own direct experience over peer voting systems, preventing traitors from hiding behind group ratings [HZN09].
- **Time Decay Factors:** Protocols give more weight to recent interactions and use statistical models, such as Bayesian updates, to detect rapid changes in node behavior [HZN09].

### 2.4.6 Orchestrated Attacks

An orchestrated attack occurs when multiple colluding entities coordinate a **multifaceted strategy** across different attack vectors, dynamically altering their behavior over time to exploit decentralized reputation systems [HZN09]. Unlike single-strategy exploits, orchestrated campaigns target not only the **formulation** of reputation metrics, but also their **calculation** and **dissemination** dimensions.

#### Mechanism and System Impact

In an orchestrated attack, adversaries divide participating identities into specialized groups to execute complementary roles simultaneously. A primary example is the **oscillation attack** [HZN09], where colluding nodes split into teams that periodically alternate between honest and dishonest behavior. Honest-behaving teams build reputation while slowing down the rating decline of dishonest teams. Once the dishonest nodes deplete their credibility, teams swap roles, allowing previously honest peers to exploit the system while the former misbehavers rebuild their scores.

Colluders evade standard detection mechanisms thanks to operating as **partially connected clusters** with indirect, loosely coupled interactions across extended time windows. By avoiding tightly connected cliques, adversaries successfully mask their structural relationships and appear to detection algorithms as independent, benign participants.

The primary system impacts of orchestrated attacks include:

- **Subversion of Aggregation Pipelines:** Controlling a significant portion of nodes responsible for score calculation and dissemination allows colluders to corrupt metric values directly within a target network region.
- **Evasion of Graph-Based Detection:** Operating through loose, indirect feedback loops allows colluding identities to evade standard clique detection algorithms while still achieving coordinated malicious goals.
- **Extended System Exploitation:** Role-swapping mechanisms continuously reset and sustain high reputation levels, enabling long-term exploitation without incurring permanent penalties.

## Mitigation Strategies

Defending against orchestrated multi-vector campaigns requires moving beyond static countermeasures and graph-clique detection:

- **Robust Metric Formulation:** Protocols design reputation functions with strict mathematical properties and asymmetry, significantly limiting the capability of malicious coalitions to manipulate scores [HZN09].
- **Adaptive Historical Weighting:** Algorithms dynamically adjust the weight of historical feedback by giving more weight to recent behavior to counter oscillatory strategies and rapidly detect nodes that turn malicious [HZN09].
- **Securing Calculation and Dissemination:** Randomizing and decoupling score calculation from dissemination pipelines prevents colluding collectives from gaining control over local aggregation processes [HZN09].

## 2.5 Discrete-Event Simulation

A Discrete-Event Simulation (DES) is a modeling method used to evaluate dynamic systems where state changes occur exclusively at **discrete, asynchronous points in time**

[UL17]. Rather than tracking continuous physical processes, a DES represents a system as a set of interconnected, attributed components whose behaviors are driven by distinct operational events. Between two consecutive events, the system state **remains completely static**.

This approach is especially suitable for modeling technical and distributed systems such as communication networks and peer-to-peer protocols where autonomous components interact through discrete transactions, state updates, and stochastic processes.

### 2.5.1 Core System Abstractions

To model both the structure and operational logic of a system, DES relies on three primary abstractions [UL17]:

- **Entities:** The individual system components (e.g., participating peers in a network) characterized by attributes that define their current state.
- **Activities:** Operations spanning a specific duration over time (e.g., processing a transaction or gathering reputation ratings).
- **Events:** Instantaneous occurrences that mark state transitions in the system (e.g., a node joining the network, issuing an interaction request, or disconnecting).

### 2.5.2 Event-Oriented Execution Mechanism

While DES models can be conceptualized through activity-oriented, process-oriented, or event-oriented worldviews, this work adopts the **event-oriented worldview** [UL17].

In this execution paradigm, simulation time does not advance continuously or in fixed clock ticks. Instead, the simulation clock jumps directly from the timestamp of the current event to the timestamp of the next scheduled event, processing the system dynamics as a sequence of discrete state snapshots.

### 2.5.3 Event Scheduling and Simulation Loop

In an event-oriented simulation, time management and execution logic are driven by a central priority queue, commonly known as the **Future Event List (FEL)** [UL17]. The FEL maintains all scheduled events ordered chronologically by their execution timestamp.

The execution engine operates through an iterative loop that proceeds as follows:

1. **Event Extraction:** The simulator pops the event with the earliest timestamp from the FEL.
2. **Clock Advance:** The simulation clock  $t_s$  jumps directly to the extracted event's timestamp, bypassing inactive intervals.
3. **State Execution:** The event routine is executed, updating entity state attributes and system variables.
4. **Event Generation:** As a result of the executed state transition, new future events may be scheduled and inserted into the FEL at their respective future timestamps.

This execution loop continues until the FEL is empty or a predefined termination condition is reached. A detailed representation of this workflow is shown in Figure 2.3.

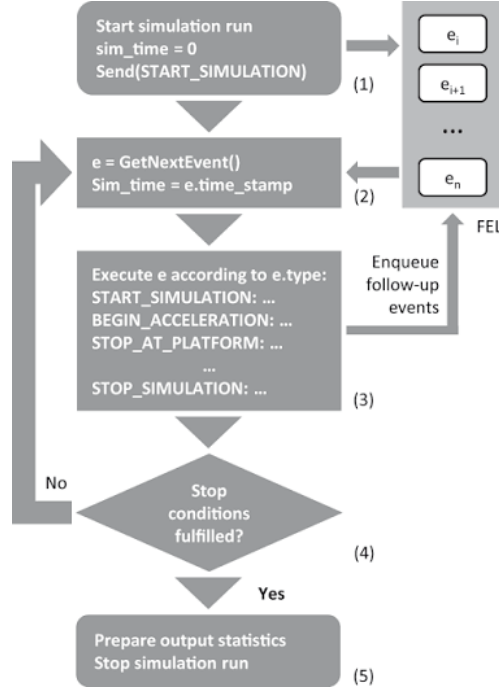


Figure 2.3: Basic event-oriented simulation execution loop Ullrich and Lückcrath [UL17].

## 2.5.4 Stochastic Modeling and Event Distributions

In real-world networks, system events such as peer arrivals, transaction requests, and network messages do not occur at fixed deterministic intervals. To capture this non-deterministic behavior, event inter-arrival times and activity durations are modeled stochastically using probability distributions.

A fundamental distribution in discrete-event simulation is the **exponential distribution**, denoted as  $\exp(\lambda)$ , where  $\lambda > 0$  represents the average rate of event occurrences per time unit [UL17]. It is used to model the continuous time elapsing between independent successive events (such as request arrivals) governed by a **Poisson** process. The probability density function (PDF) of an exponentially distributed random variable  $X$  is defined as:

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

The mean inter-arrival time  $\mu$  between consecutive events is inversely proportional to the event rate:

$$\mu = \frac{1}{\lambda}$$

Due to its memoryless property, the exponential distribution is well-suited for modeling independent peer interactions in decentralized environments, where the rate parameter  $\lambda$  defines how frequently events occur per unit time. In this setting, the underlying **Poisson distribution** models the number of events executed within a fixed time window. A lower value of  $\lambda$  ( $\lambda = 1$ ) means that fewer events take place in that window. As  $\lambda$  increases ( $\lambda = 4$  or  $10$ ), the overall number of generated events grows accordingly, making higher event counts much more likely, as can be observed in Figure 2.4.

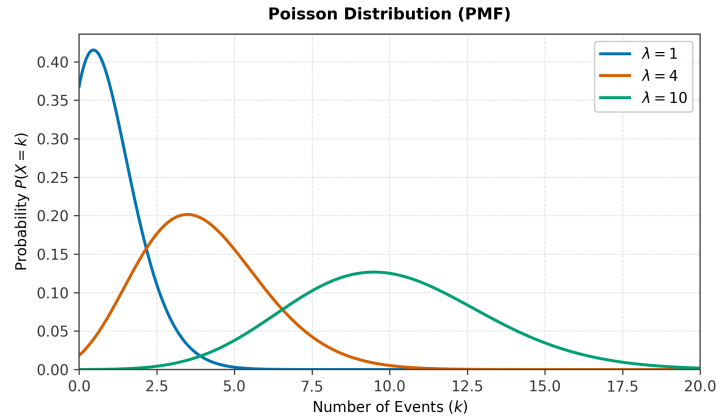


Figure 2.4: Poisson probability mass function for different values of  $\lambda$ .

While the Poisson distribution counts how many events happen within a given timeframe, the **exponential distribution** measures the time elapsed between consecutive events. A smaller  $\lambda$  ( $\lambda = 0.5$ ) means events are less frequent, resulting in longer waiting times. As  $\lambda$  increases ( $\lambda = 1.5$ ), events occur closer together, leading to shorter waiting times and a curve that drops rapidly near zero, as shown in Figure 2.5.

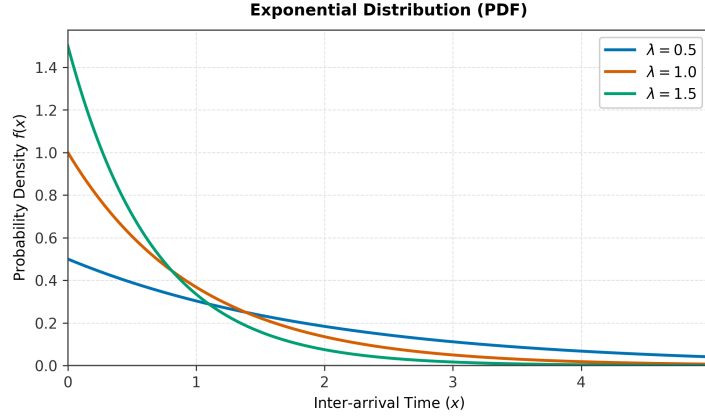


Figure 2.5: Probability density function of the exponential distribution for different values of  $\lambda$ .

### 2.5.5 Simulation Control and Data Collection

As formalized by Ullrich and Lückerrath [UL17], the execution process can be understood as a sequence of discrete **snapshots** of the model state, driven by events managed sequentially within a Future Event List priority queue.

To guarantee reproducibility across protocol evaluations, stochastic processes are governed by fixed-seed pseudo-random number generators. Throughout the simulation, performance metrics are captured using two complementary strategies:

- **Event-Driven Logging:** State transitions, such as transaction outcomes and reputation updates, are recorded dynamically inside event routines.
- **Periodic State Snapshots:** Monitoring events scheduled at fixed time intervals capture global network indicators over time.

The main execution loop repeatedly retrieves the event with the minimum timestamp from the FEL and updates the system state accordingly [UL17]. This loop iterates until reaching a predefined termination condition, such as an empty FEL or maximum simulation time.

# Chapter 3

## Methodology

Evaluating trust propagation and attack mitigation in distributed networks requires linking formal mathematical abstractions with dynamic, event-driven simulation. To achieve this, we establish a modular Discrete-Event Simulation (DES) framework designed to model system interactions, analyze routing metrics, and evaluate security countermeasures under adversarial conditions.

To start, we outline the **formal operational model** of our distributed system, specifying the mathematical rules for trust and node behavior (Section 3.1). Building on this foundation, we evaluate diverse **trust metrics** (Section 2.3), ranging from classic graph algorithms, such as shortest path distance, maximum flow, and Personalized PageRank (PPR), to advanced decentralized embedding systems introduced by Romano [Rom25] (Section 2.3.2), including Tree-BeFS, Greedy DFS with SPS, and Geometric Ping-Pong DFS. Next, we characterize various **adversarial attack vectors**, detailing how malicious nodes manipulate transaction outcomes and corrupt the global reputation matrix (Section 3.3). To counter these threats, we examine targeted **punishment mechanisms** acting as countermeasures to isolate malicious actors and preserve network security (Section 3.4). With the theoretical framework and security analysis established, we present the technical architecture of the **modular simulation project** that translates these specifications into an executable environment (Section 3.5). Finally, we highlight the **modularity and extensibility** of the framework, showing how new routing protocols, embeddings, and analytics can be seamlessly integrated into the codebase (Section 3.6).

### 3.1 Simulated System and Operational Model

Before analyzing the execution architecture, we must define the formal mathematical model of our distributed network. The system separates the long-term historical memory of a node's interactions from both the immediate computational topology and the virtual coordinate positions used for localized pathfinding.

The network state at any simulation time  $t$  is defined by the following tuple:

$$\mathcal{S}(t) = \langle \mathcal{A}(t), \mathcal{R}(t), G(t), \mathcal{L}(t) \rangle \quad (3.1)$$

where:

- $\mathcal{A}(t) \subset V$  represents the set of currently active nodes online.
- $\mathcal{R}(t)$  is the persistent global reputation matrix capturing long-term historical transaction records following the Maximization Paradigm.
- $G(t) = (\mathcal{A}(t), E(t))$  is the dynamic graph used to enforce structural trust boundaries, where  $E(t)$  is the set of active directed trust links that satisfy the threshold filtering.
- $\mathcal{L}(t)$  is the virtual embedding layout mapping each active node  $v \in \mathcal{A}(t)$  to a specific coordinate vector or multi-tree position within a decentralized space (supporting the Social Coordinate System (SoCS), multi-tree spanning structures (STGE)).

#### 3.1.1 Node Behavioral Profiles and Adversarial Model

The global set of network participants is partitioned into two disjoint subsets: honest nodes ( $V_{\text{honest}}$ ) and malicious nodes ( $V_{\text{malicious}}$ ), such that  $V = V_{\text{honest}} \cup V_{\text{malicious}}$ . When a node enters the system for the first time, its behavior is stochastically determined by a global malicious rate parameter  $\alpha \in [0, 1]$ , representing the probability  $P(v \in V_{\text{malicious}}) = \alpha$ .

- **Honest Nodes** ( $V_{\text{honest}}$ ): These nodes strictly follow the protocol rules. When acting as destinations or routing intermediaries, they consistently deliver honest and reliable service, generating legitimate transaction outcomes that reinforce mutual trust.
- **Malicious Nodes** ( $V_{\text{malicious}}$ ): These nodes seek opportunities to exploit honest peers. In the current strategy, malicious entities initially simulate honest behavior to gain trust. After completing  $k$  successful transactions, they switch to adversarial behavior and generate malicious outcomes (Section 3.3.1). However, this strategy is not fixed. Malicious behavior can change by modifying existing parameters, such as  $k$ , or by introducing new parameters into the system.

### 3.1.2 The Trust Threshold and Graph State Separation

The matrix  $\mathcal{R}(t)$  stores trust scores  $\mathcal{R}_{ij}(t) \in [0, 1]$ , where 1.0 represents absolute trust and 0.0 complete distrust. A directed edge  $e_{ij}$  exists in the active edge set  $E(t)$  of the graph  $G(t)$  if and only if both nodes are active online ( $i, j \in \mathcal{A}(t)$ ) and their trust score meets the threshold  $\theta_{\text{trust}}$ :

$$E(t) = \{(i, j) \in \mathcal{A}(t) \times \mathcal{A}(t) \mid \mathcal{R}_{ij}(t) \geq \theta_{\text{trust}}\} \quad (3.2)$$

While the network topology is determined by this threshold, the edge weight attribute in  $G(t)$  adapts to the active routing paradigm. The weight undergoes linear inversion for cost-minimizing strategies (e.g., shortest path) to map higher reputation to a lower path cost. Conversely, for score-maximizing strategies (e.g., max-flow, PPR), the reputation value is preserved.

$$\text{weight}_{ij}(t) = \begin{cases} 1.0 - \mathcal{R}_{ij}(t) & \text{if cost-minimization strategy} \\ \mathcal{R}_{ij}(t) & \text{if score-maximization strategy} \end{cases} \quad (3.3)$$

When embedding spaces are active ( $\mathcal{L}(t) \neq \emptyset$ ), routing algorithms enforce  $E(t)$  as a hard structural constraint. Because coordinate spaces are highly sensitive to adversarial manipulation, restricting pathfinding exclusively to  $E(t)$  is necessary to isolate geometric exploration from malicious nodes.

Any reputation drop below  $\theta_{\text{trust}}$  instantly prunes the edge from  $E(t)$ . If a node loses all valid incoming connections, its in-degree drops to zero ( $d_{\text{in}}(v) = 0$ ), resulting in complete **topological isolation**. This neutralizes the actor's ability to attract routing paths across both standard graph and embedding spaces.

### 3.1.3 Transaction Execution and Node Connectivity Rules

Transactions drive the chronological progression of the network state. When an active source node  $i \in \mathcal{A}(t)$  initiates a transaction, it does not evaluate the entire network; instead, it randomly samples a small subset of online nodes excluding itself, up to a fixed bound known as the candidate pool size ( $K_{\text{pool}}$ ), to construct a candidate set:

$$C_{\text{candidates}} \subseteq \mathcal{A}(t) \setminus \{i\}, \quad |C_{\text{candidates}}| = \min(|\mathcal{A}(t)| - 1, K_{\text{pool}}) \quad (3.4)$$

To determine the final target node, the source node evaluates every entity within  $C_{\text{candidates}}$  by executing its configured pathfinding routine over the active topology  $G(t)$ , utilizing the structural coordinates in  $\mathcal{L}(t)$  whenever decentralized embeddings are enabled. The node

then resolves the optimal destination  $j \in C_{\text{candidates}}$  that yields the best routing metric, minimizing path lengths for cost-based algorithms or maximizing capacities for score-based families. If no valid path to any candidate is discovered, the pathfinding attempt fails.

Upon successful path discovery and destination selection, the transaction executes. The transaction outcome is determined stochastically by sampling a discrete decimal value from the bounded set  $\mathcal{O} = \{0.0, 0.1, \dots, 1.0\}$ . The underlying selection probabilities are dynamically weighted according to the destination's behavior, which shifts toward malicious outcomes once a strategic adversary completes its structural integration threshold of exactly  $k$  successful transactions.

The lifecycle of edges within the system, from their initial creation and structural initialization to their continuous evolution, is governed by four operational states:

1. **Initial Bootstrap Connection:** When a new node joins the network for the first time, it randomly samples a baseline number of active nodes defined by the start connection parameter ( $S_{\text{conn}}$ ). It establishes immediate **bidirectional** infrastructure links with these neighbors, initializing the mutual reputation scores to a configurable value  $\mathcal{R}_{\text{init}}$  ( $\mathcal{R}_{ij} = \mathcal{R}_{\text{init}}$  and  $\mathcal{R}_{ji} = \mathcal{R}_{\text{init}}$ ).
2. **First transaction:** If a source node resolves a routing path and executes a transaction with a target node sharing no prior connection, a new directed arc is dynamically instantiated in the reputation matrix  $\mathcal{R}(t)$  upon transaction completion. The initial weight of this arc is set to the transaction outcome value.
3. **Multiple Interactions:** When a pair of nodes has prior transactions and a new one occurs, the corresponding reputation score is updated iteratively as the running average of historical transaction outcomes recorded between them.
4. **Reconnection:** When an offline node  $i$  rejoins the active set  $\mathcal{A}(t)$ , its persistent reputation scores in  $\mathcal{R}(t)$  are mapped back onto the active topology. The system reinstates its incoming and outgoing directed edges in  $G(t)$  for all active connections satisfying  $\mathcal{R}_{ij}(t) \geq \theta_{\text{trust}}$ , preserving historical reputation values without alteration. At the same time, its virtual coordinate mappings within  $\mathcal{L}(t)$  are re-synchronized.

## 3.2 Routing Algorithms Implementation

This section details the implementation and operational behavior of the routing metrics within the simulation engine, building on the theoretical foundations established in Section 2.3. We analyze how these algorithms query the network graph  $G(t)$ , apply threshold filtering, and return outputs based on their pathfinding strategy.

Pathfinding strategies are organized into two main operational paradigms: **standard graph-based metrics** operating with full global knowledge of the topology (Section 2.3.1), and **embedding-space metrics** using localized tree or continuous coordinate layouts (Section 2.3.2), adapted from the models of Romano [Rom25].

Before executing any pathfinding routine, the engine applies a **preliminary node validation**. The source node  $s$  must be active ( $s \in \mathcal{A}(t)$ ), otherwise the transaction aborts immediately. The destination target  $d$  must also be verified as active ( $d \in \mathcal{A}(t)$ ) to initiate path discovery for all algorithms except **Personalized PageRank**, which does not require a specific target node to be online because it evaluates all available nodes simultaneously from the perspective of source  $s$ .

### 3.2.1 Standard Graph-Based Metrics

When operating without an embedding system ( $\mathcal{L}(t) = \emptyset$ ), pathfinding routines execute directly over the global graph  $G(t) = (\mathcal{A}(t), E(t))$  via the **NetworkX**<sup>1</sup> library. The graph is filtered to retain only active arcs satisfying the trust threshold ( $\mathcal{R}_{uv}(t) \geq \theta_{\text{trust}}$ ) before evaluating trust metrics.

#### Shortest Path Distance

The framework computes the optimal path from source  $s$  to target  $d$  using Dijkstra’s single-source routine. When a valid connection exists, the routine returns the node sequence  $p^*$  paired with its cumulative cost  $C^*$ :  $\langle p^*, C^* \rangle$ . If the target node  $d$  is topologically isolated or disconnected, the engine catches the routing exception and returns a null tuple  $\langle \emptyset, \emptyset \rangle$ .

#### Maximum Flow

The framework computes the maximum flow from source  $s$  to target  $d$  using the Edmonds-Karp algorithm, treating edge trust scores  $c(u, v) = \mathcal{R}_{uv}(t)$  as capacities. Because the flow is evaluated across parallel channels rather than a single sequence, the routine returns an empty path paired with the total flow value  $V_f$ :  $\langle \emptyset, V_f \rangle$ . If the target node is disconnected or unreachable, the engine catches the exception and returns a null tuple  $\langle \emptyset, \emptyset \rangle$ .

---

<sup>1</sup><https://networkx.org/>

## Personalized PageRank

The framework calculates the Personalized PageRank vector with respect to source  $s$ , with edge transition weights proportional to trust scores  $\mathcal{R}_{uv}(t)$ . Unlike path-bound metrics, the routine computes centrality scores globally across all active nodes in a single run, returning an empty path paired with a score dictionary:  $\langle \emptyset, \{\text{node} : \mathbf{PR}_{\text{node}}\} \rangle$ . Executing the routine once per source node at time step  $t$  eliminates redundant per-target recalculations and minimizes computational overhead.

### 3.2.2 Embedding-Space Metrics

When operating with a decentralized layout ( $\mathcal{L}(t) \neq \emptyset$ ), pathfinding routines navigate using localized tree or continuous coordinate structures within  $\mathcal{L}(t)$  rather than performing global searches over  $G(t)$ . As in standard graph-based routines, the embedding space considers only active edges that satisfy the trust threshold ( $\mathcal{R}_{uv}(t) \geq \theta_{\text{trust}}$ ).

#### 3.2.2.1 Tree-Based Embedding Metrics

When the network layout utilizes Spanning Trees Graph Embedding (STGE), the topology maps to a set of rooted spanning trees within the layout  $\mathcal{L}(t)$ . Pathfinding routines use structural tree distances to select the optimal layout between source and target nodes, executing local routing directly across the embedded tree structures.

#### Tree Best-First Search

The framework executes a Best-First Search over the selected spanning tree in  $\mathcal{L}(t)$ , prioritizing intermediate hops based on their structural tree distance to target  $d$ . To manage search complexity, the routine maintains an active priority queue bounded by a maximum beam width parameter  $W_{\text{bef}}$ , which restricts the number of candidates retained in memory by pruning excess nodes. When a valid path is resolved, the algorithm returns the node sequence  $p$  paired with search metadata:  $\langle p, \text{res} \rangle$ . If the target is unreachable or the search fails, the engine catches the exception and returns a null tuple  $\langle \emptyset, \emptyset \rangle$ .

#### Tree Greedy Depth-First Search

The framework performs a localized depth-first exploration over  $G(t)$ , greedily selecting at each hop the neighbor that minimizes the structural tree distance to target  $d$ . The routine proceeds recursively along the most promising branch with local backtracking upon hitting

dead ends, without maintaining a global priority queue. When a valid path is resolved, the algorithm returns the node sequence  $p$  paired with search metadata:  $\langle p, \text{res} \rangle$ . If the target is unreachable or the search fails, the engine catches the exception and returns a null tuple  $\langle \emptyset, \emptyset \rangle$ .

### Tree Ping-Pong Depth-First Search

The framework executes a tree-guided depth-first search selecting the spanning tree  $T_1 \in \mathcal{L}(t)$  that minimizes the structural distance  $d_{T_1}(s, d)$ . At each step, the routine greedily expands the unvisited neighbor closest to target  $d$  according to  $d_{T_1}$ , performing local backtracking upon reaching dead ends. The search explores alternative routes up to a predefined limit, merging all discovered paths into a localized subgraph  $G_{\text{explored}} \subseteq G(t)$ . Over this bounded subgraph, the routine evaluates either Maximum Flow or Personalized PageRank. The algorithm returns an empty path paired with the computed metric:  $\langle \emptyset, V_f \rangle$  when configured with Maximum Flow, or  $\langle \emptyset, \{\text{node} : \mathbf{PR}_{\text{node}}\} \rangle$  when configured with Personalized PageRank. If the target is unreachable or the search fails, the engine catches the exception and returns a null tuple  $\langle \emptyset, \emptyset \rangle$ .

#### 3.2.2.2 Continuous Space Embedding Metrics

When the network layout utilizes the Social Coordinate System (SoCS), the topology maps to a continuous geometric space within the layout  $\mathcal{L}(t)$ . Pathfinding routines use continuous geometric distances to select optimal routes between source and target nodes, executing local decentralized routing directly across the embedded coordinate space.

### Best-First Search with NoNRS Optimization

The framework executes a Best-First Search using continuous spatial coordinates from SoCS, incorporating a density correction function  $f_{\text{NoNRS}}$  over Euclidean distances to navigate past sparse topological regions. Intermediate hops are selected from the priority queue to minimize the density-corrected spatial distance to target  $d$ . When a valid path is resolved, the algorithm returns the node sequence  $p$  paired with search metadata:  $\langle p, \text{res} \rangle$ . If the target is unreachable or the search fails, the engine catches the exception and returns a null tuple  $\langle \emptyset, \emptyset \rangle$ .

## Greedy Depth-First Search with SPS

The framework performs a continuous geometric exploration over  $G(t)$ , selecting neighboring nodes that minimize the spatial distance to target  $d$ . Once the destination is reached, the algorithm identifies the shortest path strictly within the sub-network  $G_{\text{explored}}$  uncovered during the exploration phase. When a valid path is resolved, the routine returns the optimized node sequence  $p$  paired with search metadata:  $\langle p, \text{res} \rangle$ . If the target is unreachable or the search fails, the engine catches the exception and returns a null tuple  $\langle \emptyset, \emptyset \rangle$ .

## Continuous Geometric Ping-Pong Depth-First Search

The framework executes a spatial geometric depth-first search guided by continuous coordinates. At each step, the routine greedily expands the unvisited neighbor closest to target  $d$  in the embedding space, performing local backtracking upon reaching dead ends. The exploration collects alternative routes up to a predefined node limit, merging all discovered paths into a localized subgraph  $G_{\text{explored}} \subseteq G(t)$ . Over this bounded subgraph, the routine evaluates either Maximum Flow or Personalized PageRank, returning an empty path paired with the computed metric:  $\langle \emptyset, V_f \rangle$  when configured with Maximum Flow, or  $\langle \emptyset, \{\text{node} : \mathbf{PR}_{\text{node}}\} \rangle$  when configured with Personalized PageRank. If the target is unreachable or the search fails, the engine catches the exception and returns a null tuple  $\langle \emptyset, \emptyset \rangle$ .

## 3.3 Formalization of Adversarial Attacks

Building upon the security attacks introduced in Section 2.4, adversarial attacks occur when the source node  $i$  or the selected destination node  $j$  belongs to the malicious subset  $V_{\text{malicious}} \subset V$ . Depending on the active attack configuration, these strategies either change the transaction outcome  $O$ , bounded within the operational domain  $\Omega = [O_{\min}, O_{\max}]$ , or directly update the global reputation matrix  $\mathcal{R}(t)$ .

### 3.3.1 Strategic Betrayal Attack

When a malicious node  $j \in V_{\text{malicious}}$  acts as the transaction destination, the transaction outcome  $O$  is governed by a local interaction counter  $T_{\text{counter}}(j)$  that tracks its total number of concluded transactions. The node permanently shifts from honest to adversarial behavior once the threshold parameter  $k_{\text{switch}}$  is reached:

$$O = \begin{cases} \mathcal{D}_{\text{honest}}, & \text{if } T_{\text{counter}}(j) < k_{\text{switch}} \\ \mathcal{D}_{\text{adversarial}}, & \text{if } T_{\text{counter}}(j) \geq k_{\text{switch}} \end{cases} \quad (3.5)$$

where  $\mathcal{D}_{\text{honest}}$  generates high-value outcomes that result in high reputation values within the matrix  $\mathcal{R}(t)$ , while  $\mathcal{D}_{\text{adversarial}}$  produces low-value outcomes that decrease the corresponding reputation entries in  $\mathcal{R}(t)$ .

### 3.3.2 Collusion Attack

When a malicious source node  $i \in V_{\text{malicious}}$  initiates a transaction, this strategy updates the network state through two mechanisms:

1. **Accomplice Boosting:** The source node  $i$  selects a random subset of other active malicious nodes  $M \subseteq V_{\text{malicious}} \setminus \{i\}$  up to a maximum size parameter  $k_{\text{boost}}$ . For each selected node  $m \in M$ , the source updates the reputation matrix  $\mathcal{R}(t)$  with the maximum value of 1.0, which increases their reputation entries.
2. **Malicious Destination Protection:** If the selected destination node  $j \in V_{\text{malicious}}$ , the standard generation of the trade outcome  $O$  is bypassed. The strategy forces the outcome to the maximum value:

$$O = O_{\text{max}} \quad (3.6)$$

which generates high reputation values between the colluding nodes within the matrix  $\mathcal{R}(t)$ , a visual explanation is provided in Figure 3.1.



Figure 3.1: Network state evolution during a Collusion Attack operating within the maximization paradigm. By forcing a maximum transaction outcome ( $O = 1.0$ ), malicious nodes maximize their mutual trust to reduce the risk of being isolated inside the network and increase their chances of being selected from the trade pool.

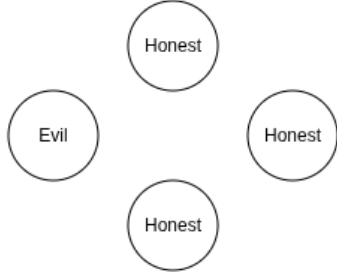
### 3.3.3 Slander Attack

When a malicious source node  $i \in V_{\text{malicious}}$  initiates a transaction with an honest destination node  $j \in V_{\text{honest}}$ , the standard generation of the trade outcome  $O$  is bypassed. The strategy forces the outcome to the minimum value:

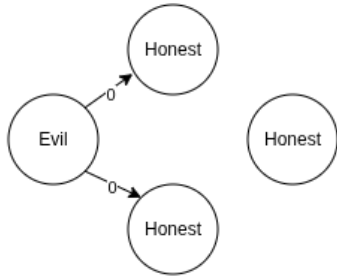
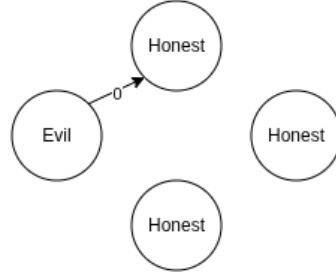
$$O = O_{\min} \quad (3.7)$$

where  $O_{\min}$  generates low-value outcomes that decrease the reputation entries of the honest node  $j$  within the matrix  $\mathcal{R}(t)$ , forcing its artificial isolation in the network topology, a visual explanation is provided in Figure 3.2.

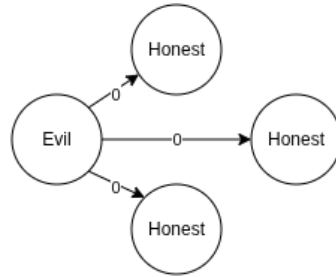
Initial state, before any Slander Transaction



After the first Slander Transaction



After the second Slander Transaction



After the third Slander Transaction

Figure 3.2: Network state evolution during a Slander Attack operating within the maximization paradigm. By forcing a minimum transaction outcome ( $O = 0$ ), the malicious source systematically drives the target's reputation arc to zero, reducing routing metrics and triggering the defense framework to isolate the honest node.

## 3.4 Countermeasures to Malicious Behavior

The mitigation framework integrates a consensus-based protocol against slander attacks with three distinct punishment mechanisms that dynamically update the reputation matrix  $\mathcal{R}(t)$  to enforce topological isolation in  $G(t)$ .

Specifically, a transaction is **classified as malicious** whenever its outcome  $O_{ij}(t)$  falls below the **default trust threshold**  $\theta_{\text{trust}} = 0.40$ , as any value below this limit could be considered a scam under the maximization paradigm. This classification updates the reputation matrix  $\mathcal{R}(t)$ , adjusting the edge weights on  $G(t)$  according to Equation (3.3). The formalization defined above is applied across all subsequent illustrative examples.

### 3.4.1 Mitigation of Slander Attacks via Localized Consensus

To prevent malicious nodes from broadcasting false alerts (Section 3.3.3) to isolate honest actors, a receiving node  $w$  validates the credibility of the alert sender  $s$  via a localized validation protocol before processing any penalty. The verification follows a two-level logic:

1. **Direct Trust Verification:** If a direct connection exists, the sender is verified as trusted if the reputation score satisfies the active paradigm criteria defined in Equation (3.3):

$$\text{Trust}(w, s) = \begin{cases} 1, & \text{if } \text{weight}_{ws}(t) \geq \theta_{\text{trust}} \\ 0, & \text{otherwise} \end{cases} \quad (3.8)$$

If  $\text{Trust}(w, s) = 1$ , the alert is processed; otherwise, it is discarded.

2. **Distributed Node Polling:** If no direct connection exists, the receiver polls its active trusted neighbors  $n \in \mathcal{A}(t)$  who share a history with the sender. A neighbor  $n$  is included in the polling pool if  $\text{Trust}(w, n) = 1$ . Each selected neighbor returns a binary opinion  $o_n \in \{0, 1\}$  corresponding to  $\text{Trust}(n, s)$ . The receiver computes the approval rate across all collected opinions:

$$\text{Approval Rate} = \frac{\sum_{n \in \text{Pool}} o_n}{|\text{Pool}|} \quad (3.9)$$

The alert is executed if and only if  $\text{Approval Rate} \geq \theta_{\text{consensus}}$  (where the default  $\theta_{\text{consensus}} = 0.66$ , following the Byzantine consensus principle). If the pool is empty, the alert is dropped.

### 3.4.2 Single Punishment

When a destination node executes a scam, the victim node **broadcasts an alert** to all active nodes in the network. However, only the **witness nodes** holding an incoming edge to the malicious actor  $m$  evaluate the notification. If the consensus-based safety protocol confirms the sender's credibility, these active witnesses  $w \in \mathcal{A}(t)$  update their reputation score  $\mathcal{R}_{wm}$ :

$$\mathcal{R}_{wm}(t+1) = \mathcal{R}_{wm}(t) \cdot \beta \quad (3.10)$$

where  $\beta = 0.5$  if the transaction outcome  $O < 0.1$ , else  $\beta = 0.7$ .

If  $\mathcal{R}_{wm}(t+1) < \theta_{\text{trust}}$ , the edge  $e_{wm}$  is pruned from the graph  $G(t)$ . Otherwise, the edge weight  $w_{wm}$  is updated based on the new reputation score  $\mathcal{R}_{wm}(t+1)$  according to Equation (3.3).

This mechanism forces the rapid exclusion of the malicious node from the topology across all compatible routing strategies, a visual explanation of the algorithm is provided in Figure 3.3.

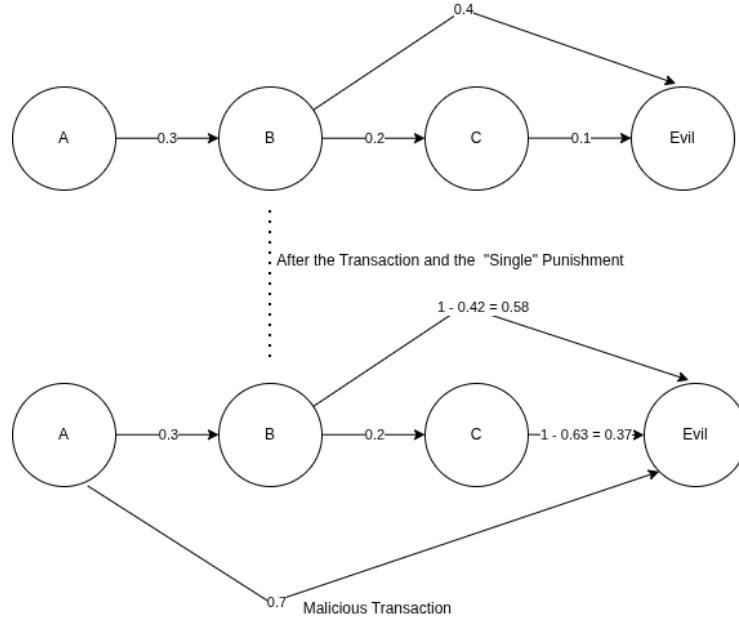


Figure 3.3: Evolution of the network state following a malicious transaction under the Single Punishment protocol operating within the cost-minimization paradigm. Upon receiving and successfully validating the broadcasted notification against the security protocol, nodes  $B$  and  $C$ , which maintain direct connections to the *Evil* node, immediately reduce and recalculate their trust value toward it.

### 3.4.3 Path Punishment

This strategy penalizes **intermediate recommenders** along the resolved routing path rather than the malicious destination. It operates exclusively under the **cost-minimization paradigm**, as this is the only paradigm that yields an explicit routing path. Unlike broadcast mechanisms, this strategy remains strictly localized and transmits no network messages. For any intermediate recommender node  $r$  located at a hop distance  $i$  from the victim node  $v$  along the path, the penalty is applied if and only if a pre-existing connection history  $\mathcal{R}_{vr}$  exists. The **penalty decays with distance**:

$$\mathcal{R}_{vr}(t+1) = \mathcal{R}_{vr}(t) \cdot \left(1.0 - \frac{0.1}{i}\right) \quad (3.11)$$

The edge is pruned from  $G(t)$  if  $\mathcal{R}_{vr}(t+1) < \theta_{\text{trust}}$ . Otherwise, the edge weight  $w_{vr}$  is updated based on  $\mathcal{R}_{vr}(t+1)$  according to Equation (3.3), a visual explanation of the algorithm is provided in Figure 3.4.

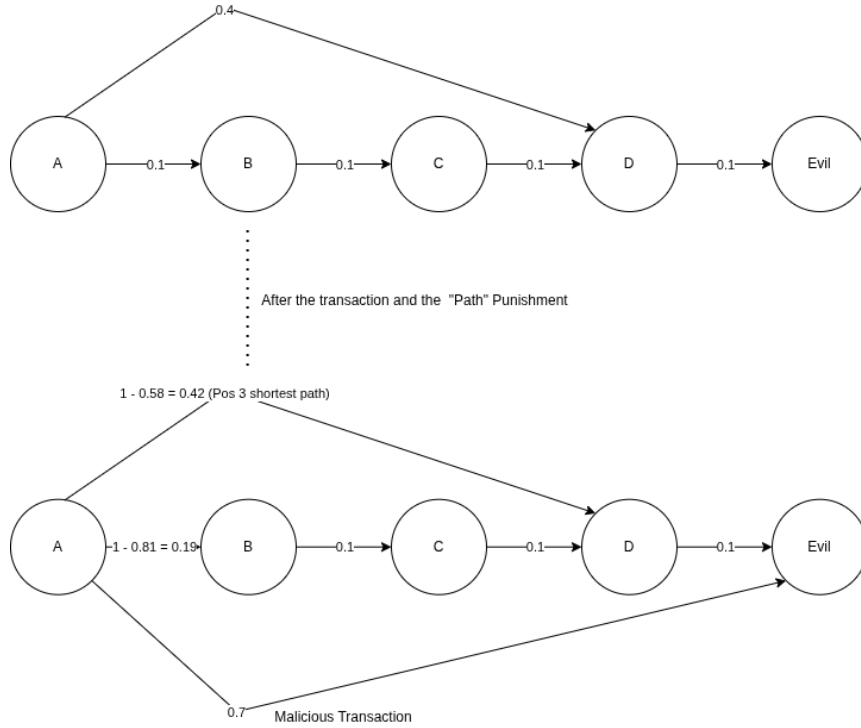


Figure 3.4: Network state evolution under the Path Punishment protocol (cost-minimization). Along the path  $A \rightarrow B \rightarrow C \rightarrow D \rightarrow Evil$ , node  $B$  receives a severe penalty due to its proximity to source  $A$ , while node  $D$  incurs a mitigated punishment, demonstrating distance-based penalty attenuation.

### 3.4.4 Recommender Punishment

This mechanism applies a penalty to the **immediate recommender**  $r$  (the node directly preceding the malicious actor in the path) and is restricted to the cost-minimization paradigm due to the requirement of an explicit sequential routing path. The victim broadcasts an alert to all active nodes in the network, but only the witnesses holding an incoming edge to  $r$  process the message. If the safety protocol validates the alert, these active witnesses  $w \in \mathcal{A}(t)$  update their reputation score toward  $r$ :

$$\mathcal{R}_{wr}(t+1) = \mathcal{R}_{wr}(t) \cdot (1.0 - 0.05) \quad (3.12)$$

The edge  $e_{wr}$  is pruned from  $G(t)$  if  $\mathcal{R}_{wr}(t+1) < \theta_{\text{trust}}$ . Otherwise, the edge weight  $w_{wr}$  is updated based on  $\mathcal{R}_{wr}(t+1)$  according to Equation (3.3), isolating recommenders that repeatedly suggest untrustworthy paths. visual explanation is provided in Figure 3.5.

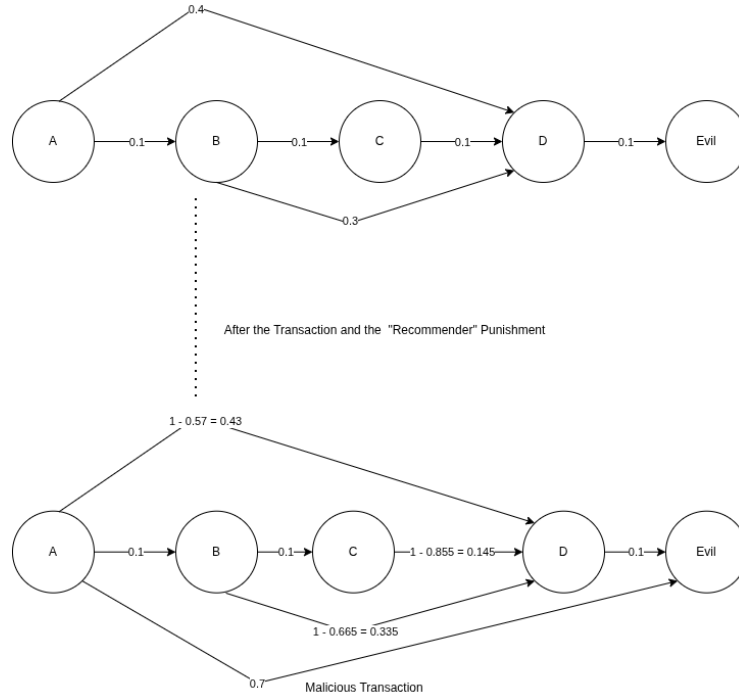


Figure 3.5: Network state evolution under the Recommender Punishment protocol (cost-minimization). Along the path  $A \rightarrow B \rightarrow C \rightarrow D \rightarrow \text{Evil}$ , the immediate recommender (node  $D$ ) incurs a penalty only from the nodes pointing to it.

## 3.5 Technical Project

Within this framework, we implemented the core discrete event simulation engine that handles time progression and event scheduling. Next, we moved forward to build the network environment component, where active node states and real-time reputation functions are executed. Lastly, we added a dedicated module for capturing, analyzing, and evaluating simulation metrics. A better understanding of the overall architecture is provided in Figure 3.6.

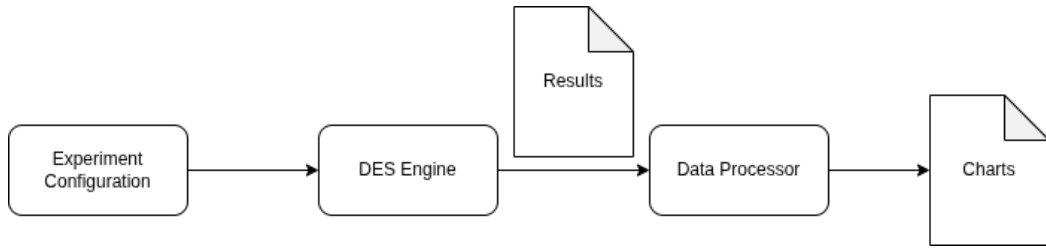


Figure 3.6: Overview of the simulation architecture. The process begins with the *Experiment Configuration*, which defines the scenario parameters for the *DES Engine*. The engine handles the time progression of the *network environment*, where node states and reputation functions are executed. The collected metrics are saved into a structured *Results* file, which is then processed by the *Data Processor* to generate the final *Charts*.

### 3.5.1 Libraries

The framework relies on three main libraries to handle graph operations, data management, and visualization.

**NetworkX**<sup>2</sup> handles all graph manipulations, maintaining the active network topology and managing the dynamic edges required by the routing algorithms.

The **pandas**<sup>3</sup> library is used to collect, organize, and structure simulation data into persistent files, supporting the decoupled data analysis phase.

Visualization is managed via **matplotlib**,<sup>4</sup> which generates charts to illustrate historical outcomes and behavioral trends.

---

<sup>2</sup><https://networkx.org/>

<sup>3</sup><https://pandas.pydata.org/>

<sup>4</sup><https://matplotlib.org/>

### 3.5.2 Custom Performance Optimization

To ensure scalability during long-running simulations over large topologies, the framework implements specific optimizations to minimize execution overhead:

- **Constant-Time Node Removal (Swap-and-Pop):** Random sampling of active nodes in  $\mathcal{O}(1)$  requires storing them in a dynamic array, utilizing Python’s native list structure. However, standard element deletion in dynamic arrays requires shifting all subsequent elements, which incurs an  $\mathcal{O}(n)$  linear time complexity. To maintain constant-time random access without this overhead, the framework pairs the list with an auxiliary hash map that tracks node positions. When a node leaves the network, it is swapped in-place with the last element of the array before performing a tail truncation. This implementation reduces the removal complexity to a constant time of  $\mathcal{O}(1)$ , preventing performance drops during high-churn scenarios.

### 3.5.3 Discrete Event Simulation Engine and System Events

The framework is built upon a Discrete Event Simulation engine. Rather than advancing time in fixed increments, the engine processes events chronologically using a priority queue for both event insertion and extraction.

The network lifecycle and node interactions are driven by five core discrete events, each updating the global network state tuple defined in Section (3.1).

#### 3.5.3.1 Network Dynamics and Churn Events

The network topology changes as nodes join, leave, and rejoin the system. These dynamics are managed by three self-scheduling events:

- **Node Join Event:** Handles the insertion of a new node. It instantiates a unique identifier, assigns a behavioral profile (honest or malicious) based on the malicious rate  $\alpha$ , and establishes the initial connections with active neighbors. It then schedules the next arrival event with an inter-arrival time sampled from an exponential distribution:

$$\Delta t \sim \text{Exp}(\lambda_{\text{join}}) \quad (3.13)$$

- **Node Leave Event:** Handles the disconnection of active nodes due to network churn. When triggered, the engine removes the target node from the active set  $\mathcal{A}(t)$  and deletes its associated edges from the graph  $G(t)$ . To handle re-entry, the event

schedules a future rejoin event for that specific node identifier with a downtime delay sampled from an exponential distribution:

$$\Delta t \sim \text{Exp}(\lambda_{\text{join}}) \quad (3.14)$$

- **Node Rejoin Event:** Handles the return of an offline node to the active network. The engine restores the node within the active set  $\mathcal{A}(t)$  and reconstructs its previous incoming and outgoing connections in the graph  $G(t)$ . To conclude, it schedules the node's next transaction and departure events using intervals sampled from their respective distributions:

$$\Delta t_{\text{trade}} \sim \text{Exp}(\lambda_{\text{trade}}), \quad \Delta t_{\text{leave}} \sim \text{Exp}(\lambda_{\text{leave}}) \quad (3.15)$$

### 3.5.3.2 Transaction Lifecycle

Node interactions are managed by a dedicated transaction event.

- **Transaction Event:** Handles trade attempts between active nodes. If at least two nodes are online, the source randomly selects a candidate pool up to a maximum size  $K_{\text{pool}}$ . The engine then attempts to find a path to these candidates using one of two algorithm families: a minimization family (e.g., Shortest Path) that evaluates the graph to minimize path lengths or distances, or a maximization family (e.g., Maximum Flow, PPR) that maximizes capacity or flow metrics.

If a valid path is found, the engine determines the transaction outcome using a weighted probability distribution. The trade is then evaluated as a scam depending on the algorithm family and a trust threshold  $\theta_{\text{trust}}$ :

$$\text{Scam} = \begin{cases} 1 - \text{Outcome} \leq \theta_{\text{trust}}, & \text{if Algorithm} \in \text{Minimization Family} \\ \text{Outcome} \leq \theta_{\text{trust}}, & \text{if Algorithm} \in \text{Maximization Family} \end{cases} \quad (3.16)$$

If a scam occurs, the framework updates the network statistics and applies a punishment protocol. To conclude, the event schedules the source node's next transaction with a delay sampled from an exponential distribution:

$$\Delta t \sim \text{Exp}(\lambda_{\text{trade}}) \quad (3.17)$$

### 3.5.3.3 System Monitoring and Structural Updates

The system executes periodic updates to synchronize geometric embeddings and log analytical state trends. A detailed discussion of the embedding mechanism is provided in Section 2.3.2.

- **Embedding Update Event:** Synchronizes the virtual coordinate layout  $\mathcal{L}(t)$  with the active graph  $G(t)$  at periodic intervals. Because continuous real-time embedding updates after every single interaction would introduce severe computational bottlenecks, this event executes on a periodic schedule to preserve simulation performance.
  - For **SoCS**, it executes high-dimensional coordinate optimization using a force-directed routine configured with dynamic node exchanges under explicit attraction and repulsion constraints.
  - For **STGE**, it builds a collection of rooted spanning trees under a specified depth dampening factor to update multi-tree spatial structures.
- **Stat Snapshot Event:** Handles data collection at periodic intervals for the future plotting, recording the real-time pathfinding success rate ( $\sigma$ ), the cumulative average scam ( $\mu$ ) before node isolation and the trusted/not trusted trades counters:

$$\sigma = \frac{\text{Total Attempts} - \text{Failed Pathfinding}}{\text{Total Attempts}} \quad (3.18)$$

$$\mu = \frac{\sum \text{Completed Scams}}{\text{Number of Malicious Records}} \quad (3.19)$$

### 3.5.4 Experiment Configuration

The simulation framework is configured using a set of parameters that control the network lifecycle, routing algorithms, and mitigation protocol. We can divide them into five main categories:

- **Routing and Algorithmic Framework:**
  - *Routing Algorithm:* Selects the pathfinding strategy executed over the active graph  $G(t)$  or within the embedding coordinate spaces (e.g., shortest path, maximum flow, Personalized PageRank, and Best-First Search).
  - *Ping-Pong Choice:* Specifies the routing metric used exclusively during the execution of Ping-Pong Choice (`ppr` or `max_flow`).
  - *Punishment Mechanism:* Defines the defense protocol triggered when a scam is detected. It can be set to `single`, `path`, `recommender`, or `none`.
- **Decentralized Embedding Spaces:**
  - *Embedding Space Type:* Selects the coordinate layout topology mapping the network, supporting the Social Coordinate System (`socs`), Spanning Trees Graph Embeddings (`stge`), or a standard graph configuration (`none`).

- *Vector Dimensionality*: Defines the mathematical dimensionality of the coordinate vectors assigned to active nodes.
- *Embedding Update Interval*: Specifies the window simulation time for periodic background updates of the node coordinates.

- **Network Churn and Dynamics:**

- *Initial Nodes*: Sets the initial network size, defining the cardinality of the active node set  $|\mathcal{A}(0)|$ .
- *Start Connections*: Determines the number of initial bidirectional bootstrap links ( $S_{\text{conn}}$ ) for a node upon joining the network.
- *Node Join Rate*: The parameter ( $\lambda_{\text{join}}$ ) of the continuous exponential distribution driving new node arrivals into the system.
- *Node Leave Rate*: The parameter ( $\lambda_{\text{leave}}$ ) of the continuous exponential distribution driving active node departures from the network.
- *Trade Rate*: The parameter ( $\lambda_{\text{trade}}$ ) of the exponential distribution governing transaction frequency per active node.

- **Reputation and Trust Mechanics:**

- *Default Reputation*: The initial trust score  $\mathcal{R}_{\text{init}}$  assigned to new bootstrap connections within the reputation matrix  $\mathcal{R}(t)$ .
- *Trust Threshold*: The filtering limit ( $\theta_{\text{trust}}$ ) used to activate or prune edges inside the graph  $G(t)$ .
- *Pool Trade Size*: The maximum candidate sampling cardinality ( $K_{\text{pool}}$ ) for transaction target selection.
- *Malicious Rate*: The global probability ( $\alpha$ ) of an entering node to be assigned as malicious ( $V_{\text{malicious}}$ ).
- *Attack Type*: Specifies the adversarial behavior strategy executed by malicious nodes (`none`, `slander`, `collusion`, or `slander_collusion`).
- *Consensus Threshold*: The minimum fraction of positive responses required from polled neighbors to achieve consensus in the defense protocol.

- **Execution and System Logging:**

- *Simulation Time*: The duration time of the simulation.
- *Snapshot Interval*: The periodic time interval for exporting structural network statistics and simulation variables for post-processing.

A summary of the simulation parameters, their corresponding command-line interface (CLI) arguments, and default values is provided in Table 3.1.

Table 3.1: Simulation Framework Parameter Reference Summary

| Parameter Description     | CLI Argument          | Symbol                      | Data Type / Choices                | Default  |
|---------------------------|-----------------------|-----------------------------|------------------------------------|----------|
| Initial Active Nodes      | --nodes               | $ \mathcal{A}(0) $          | Integer                            | 100      |
| Trust Threshold           | --threshold           | $\theta_{\text{trust}}$     | Float                              | 0.4      |
| Default Reputation        | --rep                 | $\mathcal{R}_{\text{init}}$ | Float                              | 0.5      |
| Malicious Node Rate       | --malicious           | $\alpha$                    | Float $\in [0, 1]$                 | 0.2      |
| Punishment Strategy       | --punishment          | -                           | single, path, recommender, none    | none     |
| Pool Trade Size           | --poolTrade           | $K_{\text{pool}}$           | Integer                            | 5        |
| Routing Algorithm         | --algo                | -                           | dijkstra, max_flow, ppr, befs, ... | dijkstra |
| Ping-Pong Choice          | --ping-pong-choice    | -                           | ppr, max_flow                      | ppr      |
| Simulation Time           | --time                | $max\_t$                    | Integer                            | 500      |
| Snapshot Interval         | --snap_interval       | -                           | Float                              | 5.0      |
| Start Connections         | --start_connection    | $S_{\text{conn}}$           | Integer                            | 5        |
| Node Join Rate            | --join_rate           | $\lambda_{\text{join}}$     | Float                              | 0.1      |
| Node Leave Rate           | --leave_rate          | $\lambda_{\text{leave}}$    | Float                              | 0.05     |
| Transaction Trade Rate    | --trade_rate          | $\lambda_{\text{trade}}$    | Float                              | 1.0      |
| Embedding Space Type      | --embedded_type       | -                           | socs, stge, none                   | none     |
| Embedding Dimensions      | --emb_dimensions      | -                           | Integer                            | 100      |
| Consensus Threshold       | --consensus_threshold | $\theta_{\text{consensus}}$ | Float                              | 0.66     |
| Embedding Update Interval | --embedding_interval  | -                           | Float                              | 5.0      |
| Attack Strategy Type      | --attack_type         | -                           | none, slander, collusion, ...      | none     |

### 3.5.5 Data Processor

The *Data Processor* module operates as a decoupled post-processing pipeline responsible for transforming simulation logs and metrics into analytical visualizations. As illustrated in Figure 3.7, the overall data workflow proceeds through three sequential stages:

1. **Data Ingestion:** The pipeline ingests the structured CSV files generated by the simulation engine’s logging and snapshot routines during execution.
2. **Central Orchestration:** A central **Main** controller acts as the orchestrator, loading the raw datasets, parsing metrics, and managing the execution flow.
3. **Specialized Visualization:** The **Main** controller dispatches the parsed dataset to a suite of independent plotting functions. Each function isolates a specific performance metric, such as pathfinding success rates, average scam frequencies, or topological evolution, to synthesize and export its corresponding chart artifact.

This architectural separation between data loading, orchestration, and rendering provides two primary advantages. First, it offers high modularity, allowing new plotting functions to

be added seamlessly without modifying the core data-ingestion logic. Second, it decouples visualization from execution, enabling researchers to generate arbitrary comparative charts over historical datasets without re-running computationally expensive simulation trials.

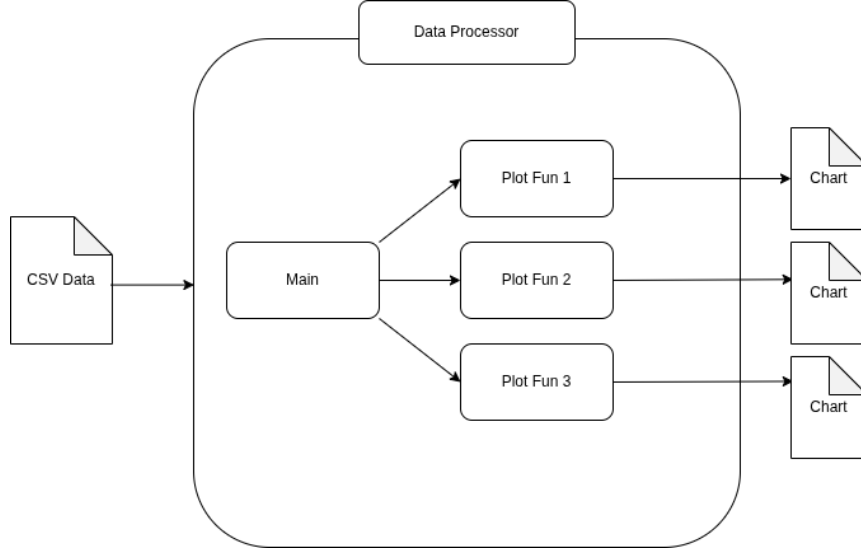


Figure 3.7: Architecture of the Data Processor module. The **Main** controller distributes the ingested CSV data to specialized plotting functions, each responsible for synthesizing a specific chart output.

## 3.6 Modularity and Extensibility

The framework is designed to integrate new functional logic through five high-level extension points, requiring developers to simply match the existing class signatures and event loops:

- **Routing Layer:** New algorithms can be integrated by extending the pathfinding interface. The routine must be classified under either the cost-minimization or capacity-maximization family, ensuring compatibility with the active paradigm’s trust evaluation logic.
- **Embedding Layer:** Custom spatial or topological layouts can be added by wrapping the calculation within the periodic embedding event. The updated node positions are written to the global state, making them immediately accessible to the search routines.

- **Attack Layer:** New adversarial strategies can be integrated by extending the attack interface. The logic intercepts the transaction execution to either alter the trade outcome  $O$  or directly update the global reputation matrix  $\mathcal{R}(t)$ .
- **Punishment Layer:** Alternative mitigation strategies can be introduced as standalone logic modules. These are triggered at the end of a transaction event, adjusting node reputation values and pruning the network topology when the predefined trust threshold is breached.
- **Analytics Layer:** New tracking metrics and charts can be integrated by capturing state data during snapshot intervals. The metrics are stored in the historical telemetry log, which updates both the dedicated plotting pipeline and the structured CSV export file.

# Chapter 4

## Results

Having defined the operational model and technical architecture of our framework in Chapter 3, this chapter presents its **empirical evaluation**. In line with the primary goals set forth in Chapter 1, we assess the **security** and **resilience** of Romano’s decentralized trust framework [Rom25] when subjected to **adversarial behaviors** in open Peer-to-Peer (P2P) environments.

To carry out this evaluation, we first establish the experimental setup, distinguishing between **static and dynamic parameters** and setting a random seed to ensure experiment reproducibility.

### 4.1 Experimental Overview

Evaluating a distributed system involves a large parameter space, where testing every combination causes a combinatorial expansion of the execution matrix ( $\prod_i m_i$ ). To keep simulation runtimes within practical limits, we split our configuration into two groups: we keep **static parameters** fixed across all simulations (e.g., threshold and default initial reputation), while changing **dynamic parameters** (e.g., malicious rate and attack patterns) across test scenarios. This setup allows us to evaluate the system’s security, resilience, and performance trade-offs without incurring excessive runtime overhead.

### 4.2 Parameter Configuration

This section details the specific values, technical justifications, and CLI flags for both static and dynamic parameters.

### 4.2.1 Static Parameters

Static parameters establish a baseline network structure and remain fixed across all simulations to isolate the impact of dynamic variables. The baseline values were determined through preliminary empirical tests to balance computational efficiency, network stability, and security:

- **Initial Active Nodes (100) and Simulation Time (500):** Establishes a controlled, small-to-moderate network scale and timeline to observe long-term routing dynamics without incurring excessive computational overhead from oversized graphs or event queues.
- **Default Reputation (0.5) and Trust Threshold (0.4):** Assigns a neutral baseline reputation to new nodes upon entry to prevent premature exclusion, while setting a minimum trust threshold required to identify and drop untrusted peers.
- **Start Connections (5):** Restricts initial node connectivity to limit attacker maneuverability, preventing malicious peers from exploiting sparse topologies or gaining excessive direct access to the network.
- **Transaction Trade Rate (1.0) and Node Join Rate (0.1):** Regulates baseline network activity by ensuring transaction frequency significantly exceeds node entry events.
- **Snapshot Interval (5.0) and Embedding Update Interval (5.0):** Synchronizes metric logging with graph updates to maintain high temporal data resolution while bounding the execution overhead over long simulation runs.
- **Embedding Dimensions (100) and Consensus Threshold (0.66):** Fixes the vector space size to match Romano’s reference architecture, while adopting the standard Byzantine Fault Tolerance criterion (2/3 agreement) for alert consensus propagation.

Table 4.1 details the static parameter configuration maintained across all simulation runs.

Table 4.1: Static Parameter Configuration Across All Simulations

| Parameter Description     | CLI Argument                       | Data Type | Fixed Value |
|---------------------------|------------------------------------|-----------|-------------|
| Initial Active Nodes      | <code>--nodes</code>               | Integer   | 100         |
| Simulation Time           | <code>--time</code>                | Integer   | 500         |
| Default Reputation        | <code>--rep</code>                 | Float     | 0.5         |
| Trust Threshold           | <code>--threshold</code>           | Float     | 0.4         |
| Start Connections         | <code>--start_connection</code>    | Integer   | 5           |
| Transaction Trade Rate    | <code>--trade_rate</code>          | Float     | 1.0         |
| Node Join Rate            | <code>--join_rate</code>           | Float     | 0.1         |
| Snapshot Interval         | <code>--snap_interval</code>       | Float     | 5.0         |
| Embedding Update Interval | <code>--embedding_interval</code>  | Float     | 5.0         |
| Embedding Dimensions      | <code>--emb_dimensions</code>      | Integer   | 100         |
| Consensus Threshold       | <code>--consensus_threshold</code> | Float     | 0.66        |

## 4.2.2 Dynamic Parameters

Dynamic parameters represent the core experimental variables modified across simulation runs to stress-test the network. These variables directly evaluate system behavior, attack resilience, and routing performance under the operational conditions defined in Chapter 3:

- **Embedding Space Type** (`socs`, `stge`, `none`): Defines the macro topology of the network. When set to `none`, routing relies on standard graph-based metrics (Section 3.2.1); otherwise, geometric or tree-based coordinate embeddings are applied (Section 3.2.2).
- **Node Leave Rate** ( $\lambda_{\text{leave}}$ ): Toggled between standard node churn (0.05) in non-embedded scenarios and near-zero churn ( $10^{-9}$ ) in coordinate-based scenarios to prevent embedding space instability during graph updates.
- **Pool Trade Size** (1, 5): Defines the number of candidate peers sampled during a trade partner search to evaluate how system dynamics and security adapt relative to the volume of top-ranked peers selected via optimal metrics.
- **Malicious Node Rate** ( $\alpha$ ): Increases progressively (0.2, 0.3, 0.4, 0.51) to evaluate network resilience under varying adversarial densities both with and without active attacks, measuring how supporting punishment strategies mitigate performance decay up to a malicious majority.
- **Attack Strategy Type** (`slander`, `collusion`, `slander_collusion`, `none`): Models the threat scenarios formalised in Section 3.3.

- **Punishment Strategy** (`single`, `path`, `recommender`, `none`): Evaluates the countermeasures described in Section 3.4.
- **Routing Algorithm** (`dijkstra`, `max_flow`, `ppr`, `befs`, `befs_nonrs`, `greedy_dfs`, `greedy_dfs_sps`, `ping_pong_dfs`): Evaluates the routing metrics defined in Section 3.2.
- **Ping-Pong Choice** (`ppr`, `max_flow`): Defines the underlying routing metric used when executing the `ping_pong_dfs` routing option.

Table 4.2 summarizes the dynamic parameter configuration used across the evaluated scenarios.

Table 4.2: Dynamic Parameter Configuration Across Evaluated Scenarios

| Parameter Description | CLI Argument                    | Data Type / Choices                                                                                                                                                                                       |
|-----------------------|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Embedding Space Type  | <code>--embedded_type</code>    | <code>socs</code> , <code>stge</code> , <code>none</code>                                                                                                                                                 |
| Node Leave Rate       | <code>--leave_rate</code>       | Float $\in \{0.05, 10^{-9}\}$                                                                                                                                                                             |
| Pool Trade Size       | <code>--poolTrade</code>        | Integer $\in \{1, 5\}$                                                                                                                                                                                    |
| Malicious Node Rate   | <code>--malicious</code>        | Float $\in \{0.2, 0.3, 0.4, 0.51\}$                                                                                                                                                                       |
| Attack Strategy Type  | <code>--attack_type</code>      | <code>slander</code> , <code>collusion</code> , <code>slander_collusion</code> , <code>none</code>                                                                                                        |
| Punishment Strategy   | <code>--punishment</code>       | <code>single</code> , <code>path</code> , <code>recommender</code> , <code>none</code>                                                                                                                    |
| Routing Algorithm     | <code>--algo</code>             | <code>dijkstra</code> , <code>max_flow</code> , <code>ppr</code> , <code>befs</code> ,<br><code>befs_nonrs</code> , <code>greedy_dfs</code> , <code>greedy_dfs_sps</code> ,<br><code>ping_pong_dfs</code> |
| Ping-Pong Choice      | <code>--ping_pong_choice</code> | <code>ppr</code> , <code>max_flow</code>                                                                                                                                                                  |

### 4.3 System Behavior Under Attack and Countermeasures

This section evaluates system performance under active threat scenarios. We first examine how the routing algorithms implemented in Section 3.2 perform against the adversarial attack patterns formalized in Section 3.3, and subsequently evaluate the impact of applying the defense mechanisms defined in Section 3.4. The evaluation is conducted through a two-step analysis, which together provide a comprehensive empirical understanding of how both the metrics and the network behave under stress. First, we examine the temporal dynamics of the network using graphical plots to track system behavior as the simulation progresses. Second, we analyze end-of-simulation transaction data, focusing on how honest

and malicious nodes participate as source or destination, and how often both honest and malicious nodes are cheated during transactions.

To maintain a concise evaluation while avoiding redundant observations, the analysis focuses on two boundary threat levels: a low-stress baseline ( $\alpha = 0.20$ ) and an majority scenario ( $\alpha = 0.51$ ) where malicious nodes outnumber honest peers. Across all evaluations, the behavior switch threshold (Section 3.1.1) remains fixed at 5 successful transactions.

To evaluate network stability, security, and routing resilience across these scenarios, we use three primary metrics:

- **Cumulative Honest Transaction Rate ( $T_{\text{honest}}$ ):** Tracks the proportion of honest transactions completed in the network over time  $t$ . Its temporal trend illustrates system behavior under attack, reflecting how rapidly active countermeasures facilitate system recovery. It is calculated as:

$$T_{\text{honest}}(t) = \frac{\text{Honest Transactions}(t)}{\text{Total Transactions}(t)} \times 100 \quad (4.1)$$

- **Pathfinding Success Rate ( $P_{\text{success}}$ ):** Measures the ability of routing algorithms to find valid paths between source and destination peers, covering transactions among both honest and malicious nodes. This metric evaluates whether network connectivity is degraded by attack strategies or active defense mechanisms (e.g., node evictions), highlighting embedding schemes that struggle to maintain reachability under stress. It is defined as:

$$P_{\text{success}} = \frac{\text{Total Attempts} - \text{Failed Pathfinding}}{\text{Total Attempts}} \quad (4.2)$$

- **Average Scams Before Isolation ( $S_{\text{avg}}$ ):** Quantifies how many successful scams a malicious node completes on average before the network isolates it. This metric evaluates the speed and strength of our punishment strategies in identifying and removing dishonest peers from the network, calculated as:

$$S_{\text{avg}} = \frac{\sum_{i \in \mathcal{I}} \text{Scams}_i}{|\mathcal{I}|} \quad (4.3)$$

where  $\mathcal{I}$  represents the set of isolated malicious nodes, and  $\text{Scams}_i$  is the number of scams committed by node  $i$  at the time of its isolation.

## Cross-Metric Interpretation

When evaluated in isolation, individual metrics may offer an incomplete or potentially misleading view of network health under certain attack conditions. Their true analytical value emerges when examining them concurrently.

For instance, having a high Cumulative Honest Transaction Rate  $T_{\text{honest}}$  along with a low Pathfinding Success Rate  $P_{\text{success}}$  shows that the total number of completed transactions has dropped significantly. While the few successful transactions are mostly honest, the network is actually failing to route most requests. This pattern could point to network disconnections or malicious peers positioned as key bottlenecks to block traffic. Therefore, what looks like a strong security score in  $T_{\text{honest}}$  might actually hide a major loss in network capacity and availability, giving a false sense of safety while the system is heavily slowed down.

Another key example is observing a high  $T_{\text{honest}}$  alongside a high or static number of scams before isolation. This combination indicates that malicious nodes are rarely evicted from the system, meaning a high  $T_{\text{honest}}$  can actually be misleading particularly when strong mechanisms like single punishment interact with slander-collusion attacks.

For instance, consider a slander-collusion attack under single punishment with a candidate pool size of 5. Malicious nodes assign the lowest possible rating (0) to honest peers while boosting each other’s reputation through collusion. Because malicious nodes initially behave well before switching strategies, they retain high reputation scores and can partially bypass the defense protocol defined in Section 3.4.1. This allows them to turn single punishment against honest peers, lowering their overall reputation scores and gaining greater control over the network.

As a result, when the system samples candidate peers to find top trade partners, malicious nodes almost always occupy the highest ranks. This creates a starvation scenario for honest peers, who are rarely selected as destination nodes, making honest-to-honest transactions nearly impossible. This reinforces our main point: reading a high  $T_{\text{honest}}$  alone can be misleading, as ongoing collusive transactions between attackers hide the starvation of honest peers. Evaluating these metrics together is therefore essential to capture the real behavior of the network.

### 4.3.1 Normal Run Scenario

To establish a performance baseline, the system is evaluated under standard operational conditions, excluding specialized attack strategies (`--attack_type none`) and active punishment mechanisms (`--punishment none`). In this setup, the candidate pool size is restricted to 1 (`--poolTrade 1`), forcing source nodes to evaluate candidate peers strictly one at a time.

Figures 4.1 and 4.2 illustrate the temporal evolution of  $T_{\text{honest}}$  and  $P_{\text{success}}$  across the evaluated malicious node ratios ( $\alpha = 0.20$  and  $\alpha = 0.51$ , respectively). Note that the average scam metric  $S_{\text{avg}}$  is deliberately omitted from these plots, as no countermeasures are deployed in this baseline setup. The same convention applies to all subsequent experiments

conducted without countermeasures.

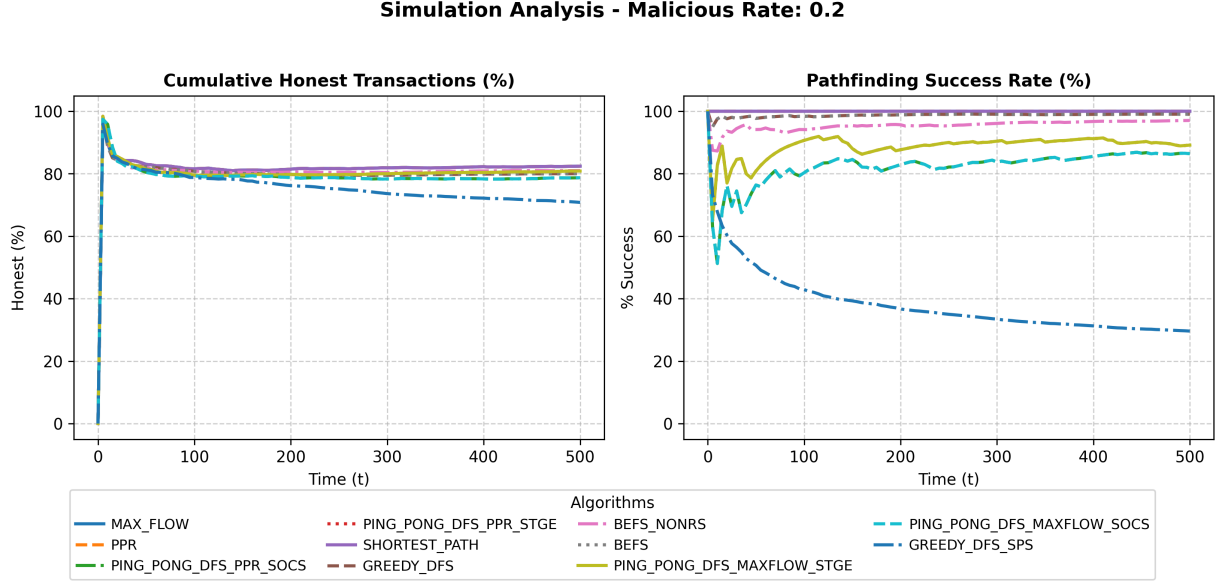


Figure 4.1: Performance of routing algorithms under baseline conditions ( $\alpha = 0.20$ , pool trade size = 1), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

As observed across the plots, the Cumulative Honest Transaction Rate  $T_{\text{honest}}$  consistently decreases as the malicious rate rises from  $\alpha = 0.20$  to  $\alpha = 0.51$ . In the absence of targeted attack strategies and countermeasures,  $T_{\text{honest}}$  scales almost directly in proportion to the honest-to-malicious node ratio across nearly all analyzed routing algorithms.

A notable exception is **GREEDY\_DFS\_SPS**, which exhibits the worst overall performance in the network across both scenarios, leaving its underlying cause as an open question. This degradation is equally evident in the Pathfinding Success Rate  $P_{\text{success}}$ , where **GREEDY\_DFS\_SPS** is the sole approach dropping below 40% over time in both configurations. This preliminary insight highlights the inherent instability of **GREEDY\_DFS\_SPS**, indicating that it struggles even when malicious nodes operate passively without targeted attack vectors, making it a primary candidate for an insecure routing mechanism.

Conversely, all other evaluated algorithms demonstrate strong resilience in pathfinding, maintaining  $P_{\text{success}}$  above an 80% threshold across both scenarios. This level of pathfinding stability is particularly notable given the small scale of the simulated network topology, where even a small number of untrusted nodes can easily sever critical bridging paths.

### Simulation Analysis - Malicious Rate: 0.51

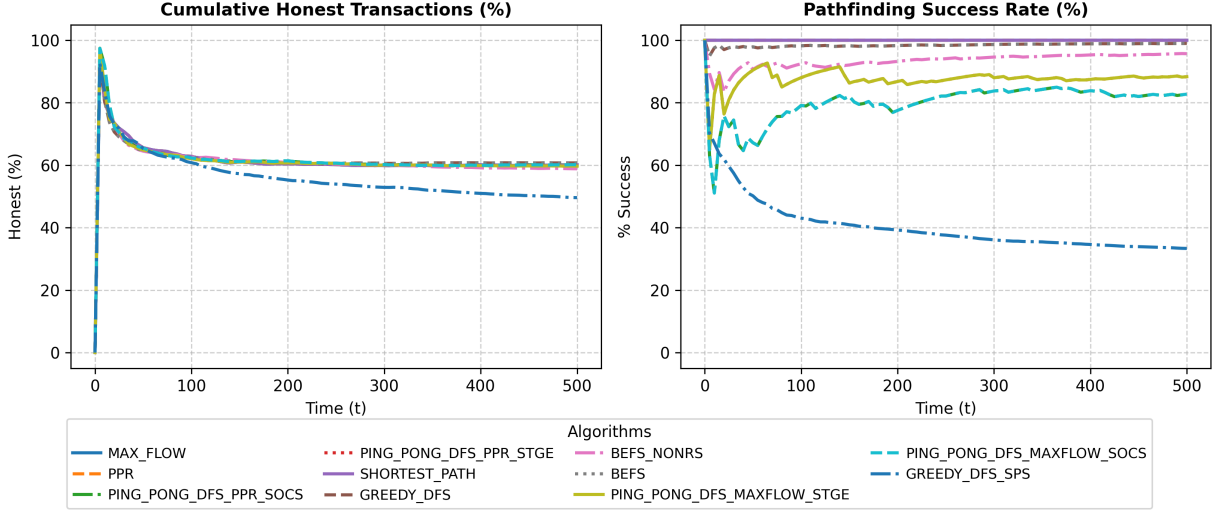


Figure 4.2: Performance of routing algorithms under majority adversarial control ( $\alpha = 0.51$ , pool trade size = 1), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

### 4.3.2 Slander Attack Scenario

For this test case, we evaluate the slander attack introduced in Section 3.3.3. The evaluation maintains the same configuration as the normal run scenario, with active punishment mechanisms disabled (`--punishment none`) and candidate pool size set to 1 (`--poolTrade 1`), modifying only the attack parameter (`--attack_type slander`).

Figures 4.3 and 4.4 illustrate the temporal evolution of  $T_{\text{honest}}$  and  $P_{\text{success}}$  across the evaluated malicious node ratios ( $\alpha = 0.20$  and  $\alpha = 0.51$ , respectively).

Regarding the Cumulative Honest Transaction Rate  $T_{\text{honest}}$ , a distinct performance degradation is observed across both malicious node ratios compared to the normal run scenario analyzed in Section 4.3.1. Specifically, negative transactions increase by approximately 20% system-wide. This highlights a major security vulnerability inherent to slander attacks across all evaluated routing algorithms, as accumulated false negative ratings artificially lower the selection probability of honest nodes.

In contrast, the Pathfinding Success Rate  $P_{\text{success}}$  percentages remain virtually unchanged relative to the Normal Run. This demonstrates that the slander attack does not impact path resolution, indicating that all evaluated algorithms maintain robustness against this specific attack vector in terms of discovering valid routes.

Overall, these results show that slander attacks do not disrupt network connectivity, but

### Simulation Analysis - Malicious Rate: 0.2

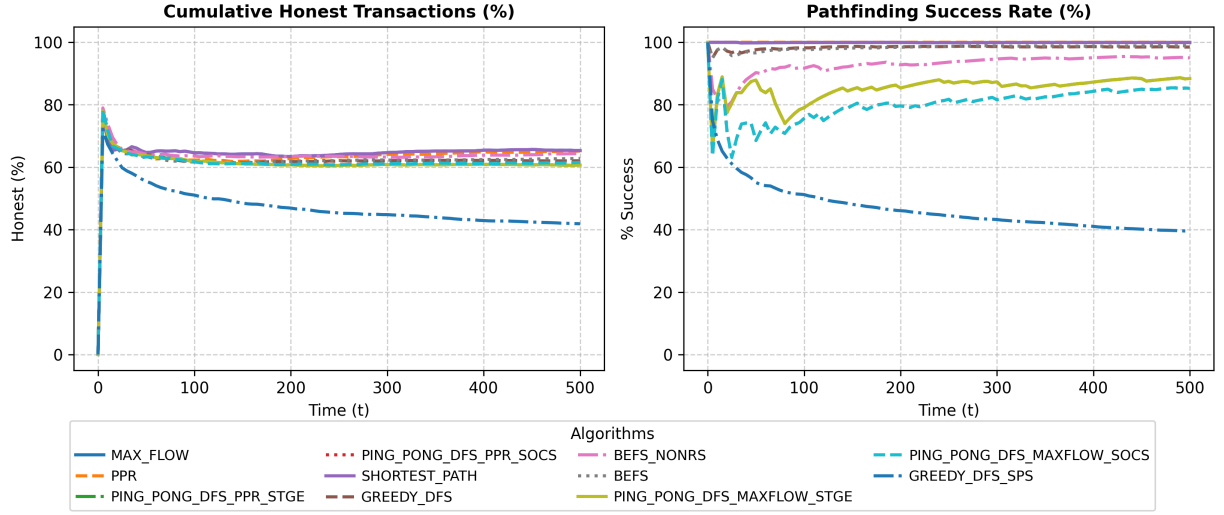


Figure 4.3: Performance of routing algorithms under Slander attack ( $\alpha = 0.20$ , pool trade size = 1), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

### Simulation Analysis - Malicious Rate: 0.51

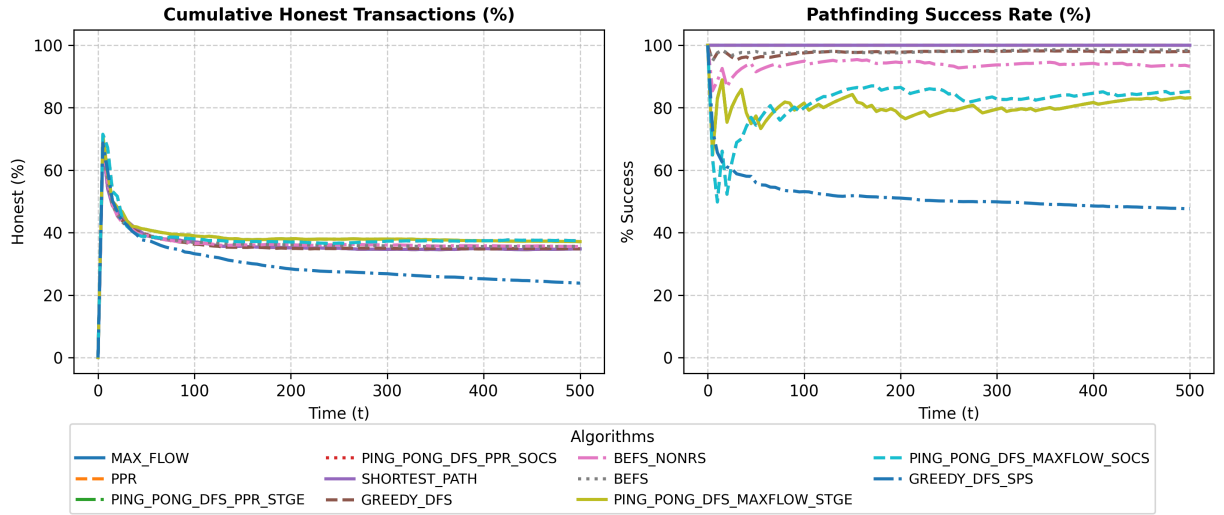


Figure 4.4: Performance of routing algorithms under Slander attack ( $\alpha = 0.51$ , pool trade size = 1), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

rather degrade transactional trust. In this configuration, using a pool trade size of 1 limits the impact on final node selection, as the metric retrieved for a node is not compared against competing candidates. Consequently, the attack would be expected to affect only path rating by reducing the number of safe paths due to negative reviews. However, this path reduction does not occur. Nevertheless, the situation differs for an external observer, as the attack introduces substantial confusion in identifying whether nodes are truly honest or malicious, since both honest and malicious nodes execute honest transactions. Specifically, malicious nodes operate honestly during their initial phase and when interacting with other malicious peers, while honest nodes do so whenever trading with other honest participants, and both groups simultaneously receive constant negative ratings.

### 4.3.3 Collusion Attack Scenario

For this test case, we evaluate the collusion attack strategy introduced in Section 3.3.2. The configuration remains the same as the normal run scenario described in Section 4.3.1, modifying exclusively the attack parameter (`--attack_type collusion`).

Figures 4.5 and 4.6 illustrate the temporal progression of  $T_{\text{honest}}$  and  $P_{\text{success}}$  across the evaluated malicious node ratios ( $\alpha = 0.20$  and  $\alpha = 0.51$ , respectively).

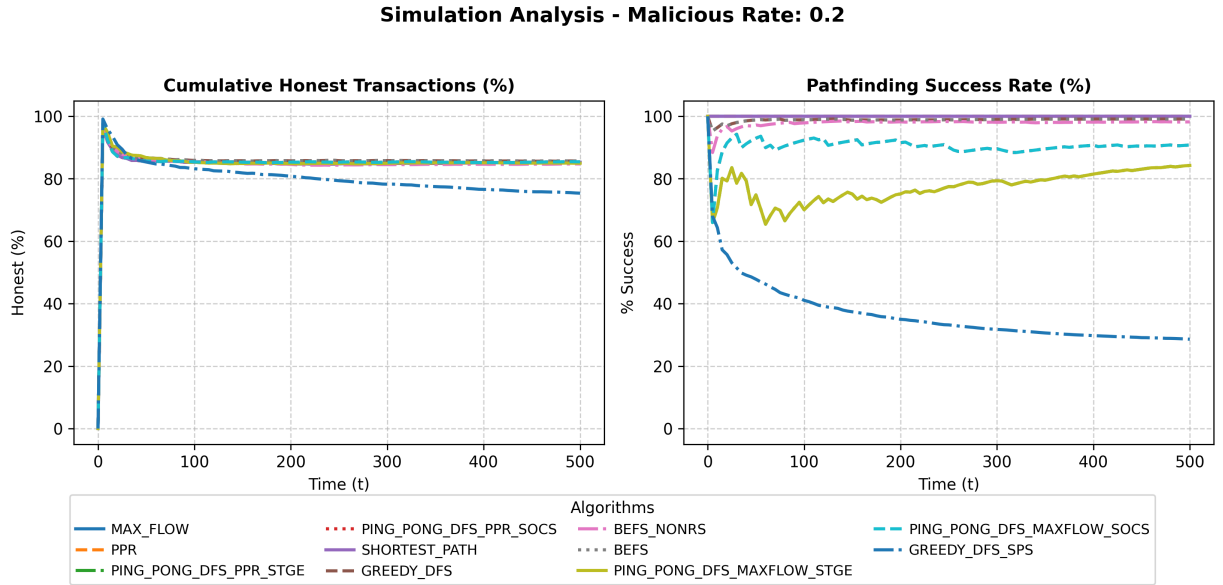


Figure 4.5: Performance of routing algorithms under Collusion attack ( $\alpha = 0.20$ , pool trade size = 1), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

Looking at the Cumulative Honest Transaction Rate  $T_{\text{honest}}$ , collusion artificially inflates

### Simulation Analysis - Malicious Rate: 0.51

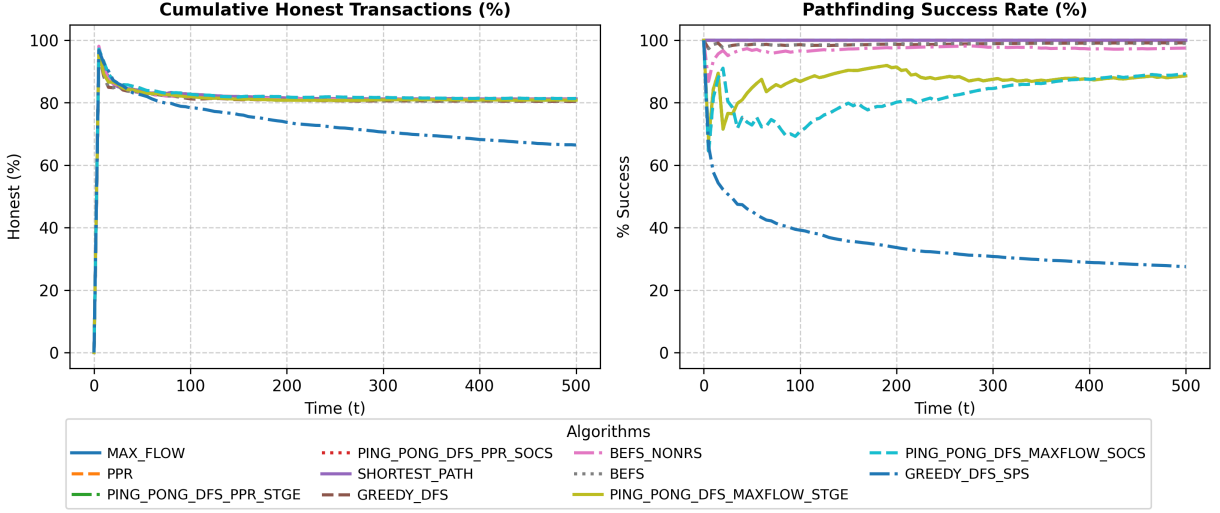


Figure 4.6: Performance of routing algorithms under Collusion attack ( $\alpha = 0.51$ , pool trade size = 1), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

overall system metrics. This inflation directly stems from the attack mechanism, as malicious nodes mutually boost each other through internal trades. Across all evaluated malicious node ratios,  $T_{\text{honest}}$  remains stable at around 80%, representing a noticeable increase compared to the normal run scenario (Section 4.3.1). These internal collusive trades effectively mask the attack, generating high global metrics that project a false impression of network health while honest nodes are systematically exploited in the background.

Regarding the Pathfinding Success Rate  $P_{\text{success}}$ , the algorithms find more paths compared to the baseline run in Section 4.3.1. Although this higher path count might seem beneficial on paper due to increased network density, these discovered routes are actually unsafe, as they heavily cross malicious nodes that cooperate among themselves.

An interesting exception appears in PING\_PONG\_MAX\_FLOW, where embedding behavior reverses compared to the normal run scenario (Section 4.3.1). Here, **stge** achieves lower path resolution than **socs**, diverging from the usual trend. This could suggest that **stge** partially resists the collusion attack by avoiding compromised routes. However, this defensive effect disappears at  $\alpha = 0.51$ , where **stge** aligns with **socs** as both metrics exhibit an upward trend over time.

As observed across all test scenarios, GREEDY\_DFS\_SPS remains the algorithm most vulnerable to adversarial activity. Ultimately, collusion introduces major security risks because its high performance metrics mask other potential attacks. This prevents defensive coun-

termes from triggering, as the system detects no operational anomalies within the network across all analyzed metrics.

#### 4.3.4 Slander-Collusion Attack Scenario

For this last case, we evaluate the combined slander-collusion attack strategy, which merges the mechanisms of the slander and collusion attacks previously described. The configuration remains the same as the normal run scenario described in Section 4.3.1, modifying exclusively the attack parameter (`--attack_type slander_collusion`).

Figures 4.7 and 4.8 show  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time for malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$ , respectively.

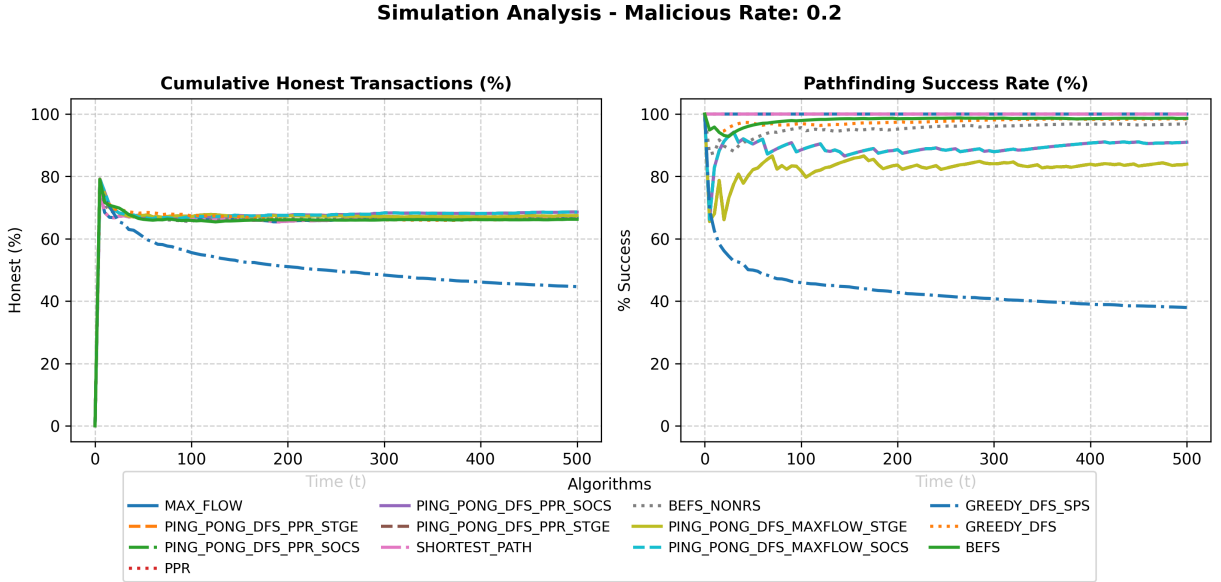


Figure 4.7: Performance of routing algorithms under Slander-Collusion attack ( $\alpha = 0.20$ , pool trade size = 1), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

Looking at the Cumulative Honest Transaction Rate  $T_{\text{honest}}$ , values are higher than in the pure slander scenario (Section 4.3.2) because colluding peers generate artificial positive feedback among themselves. While this higher rate might superficially look like performance recovery, it actually masks the attack and complicates anomaly detection. Consequently, honest nodes remain vulnerable to exploitation while malicious peers operate undetected. Among all evaluated algorithms, **GREEDY\_DFS\_SPS** suffers the most under this combined scenario, showing a sharp increase in negative transactions compared to all previous test cases.

### Simulation Analysis - Malicious Rate: 0.51

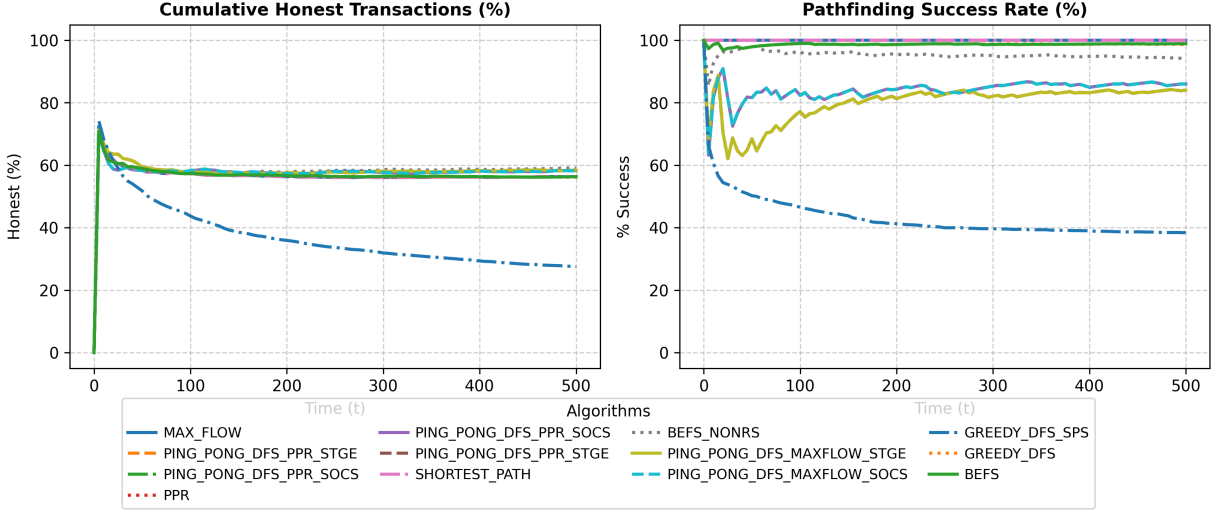


Figure 4.8: Performance of routing algorithms under Slander-Collusion attack ( $\alpha = 0.51$ , pool trade size = 1), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

Regarding the Pathfinding Success Rate  $P_{\text{success}}$ , performance closely mirrors the pure collusion scenario (Section 4.3.3). Since slander attacks do not alter path resolution, route discovery dynamics remain entirely governed by the collusive behavior of adversarial nodes.

Overall, this combined strategy represents the most potent attack vector evaluated, posing the greatest threat to network stability. By uniting collusion and slander, the attack maximizes stealth. Positive transaction rates increase relative to pure slander, while malicious nodes retain their capacity to exploit honest participants. Furthermore, sustained pathfinding efficiency keeps adversarial nodes active in routing paths. This allows other malicious peers to be selected because they maintain high mutual reputation scores, making anomaly detection particularly complex across all malicious rates ( $\alpha$ ). Consequently, the network faces a covert threat that compromises transactional trust because the combined masking effect prevents an accurate estimation of the actual number of malicious nodes in real-world environments, where any participant could potentially be dishonest. This issue persists across all analyzed metrics while causing major performance degradation in GREEDY\_DFS\_SPS, establishing it as the least resilient algorithm among all evaluated options.

### 4.3.5 Trade Pooling Defense

To evaluate candidate pool selection as a passive defense mechanism, we examine system performance across the previously analyzed attack scenarios by increasing the pool size to 5 (`--poolTrade 5`). In this setup, source nodes sample up to 5 candidate peers and select the one with the highest reputation score. Active punishment mechanisms remain disabled (`--punishment none`) to isolate the specific impact of candidate pooling. This preliminary setup provides initial insights that serve as a foundation for the comprehensive transaction analysis conducted in Section 4.3.9.1.

#### 4.3.5.1 Normal Run Scenario

This section evaluates candidate pooling under the normal run scenario. Figures 4.9 and 4.10 present the performance metrics  $T_{\text{honest}}$  and  $P_{\text{success}}$  for a candidate pool size of 5 across malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$ , respectively.

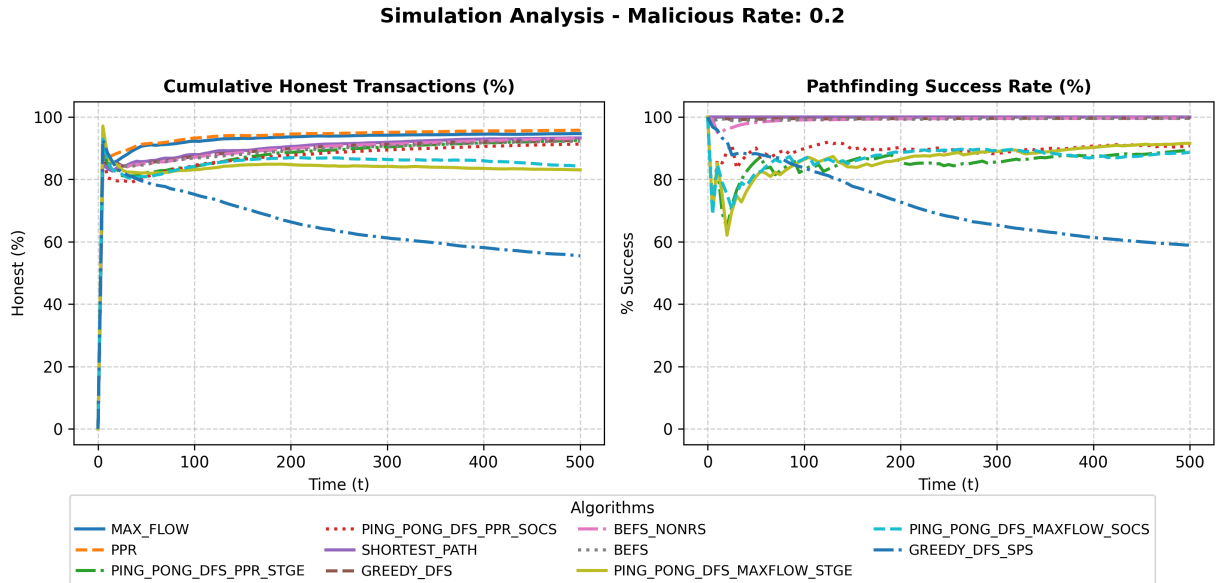


Figure 4.9: Performance of routing algorithms under the Normal Run scenario with candidate pool size = 5 ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

Regarding the Cumulative Honest Transaction Rate  $T_{\text{honest}}$ , candidate pooling yields a substantial performance improvement across all malicious rates compared to the scenario in Section 4.3.2. At lower malicious levels ( $\alpha = 0.20$ ), metrics show a general increase of approximately 10%. The most notable impact, however, occurs at the higher malicious rate ( $\alpha = 0.51$ ), where nearly all algorithms experience a remarkable 20% boost in performance.

### Simulation Analysis - Malicious Rate: 0.51

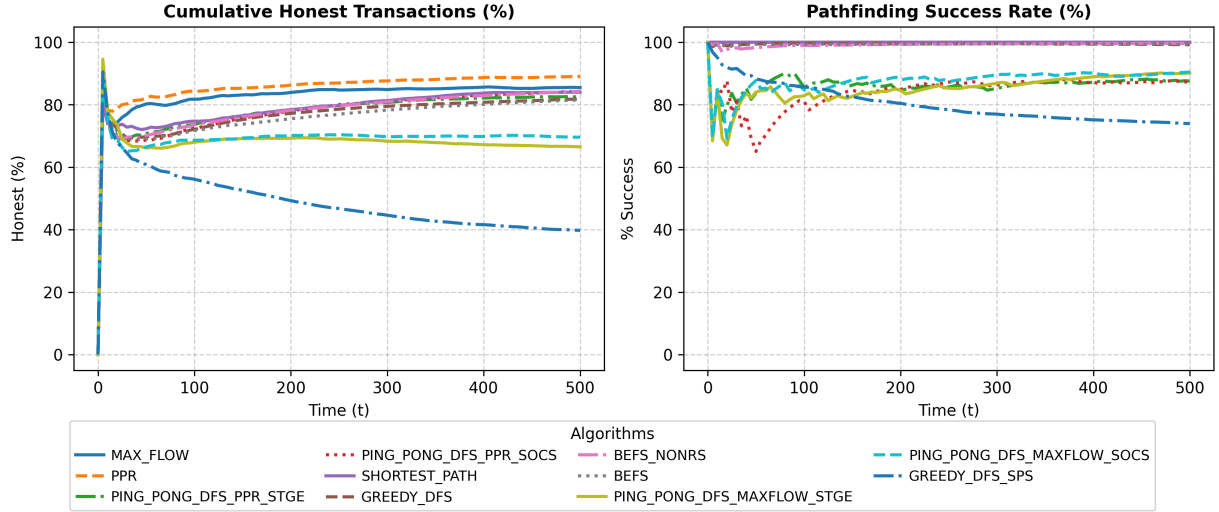


Figure 4.10: Performance of routing algorithms under the Normal Run scenario with candidate pool size = 5 ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

The sole exception to this trend is **GREEDY\_DFS\_SPS**, which once again fails to leverage the candidate pool effectively.

For the Pathfinding Success Rate  $P_{\text{success}}$ , candidate pooling once again demonstrates significant benefits. Evaluating a candidate pool of five nodes expands the available routing choices, allowing the algorithm to select the optimal destination among them. Consequently, the probability of failing to establish a path or encountering an unsafe route within these five available candidates drops substantially.

Overall, candidate pooling proves to be a highly effective enhancement in an attack-free setting. Across both low and high malicious rates, evaluating multiple top-ranked candidates increases the likelihood that selected destinations are genuinely honest nodes, thereby naturally maximizing both successful positive transactions and Pathfinding Success Rates in the absence of targeted adversarial manipulation.

#### 4.3.5.2 Slander Attack Scenario

This section evaluates candidate pooling under the slander attack scenario. Figures 4.11 and 4.12 present the performance metrics  $T_{\text{honest}}$  and  $P_{\text{success}}$  for a candidate pool size of 5 across malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$ , respectively.

Regarding the Cumulative Honest Transaction Rate  $T_{\text{honest}}$ , a significant improvement in

### Simulation Analysis - Malicious Rate: 0.2

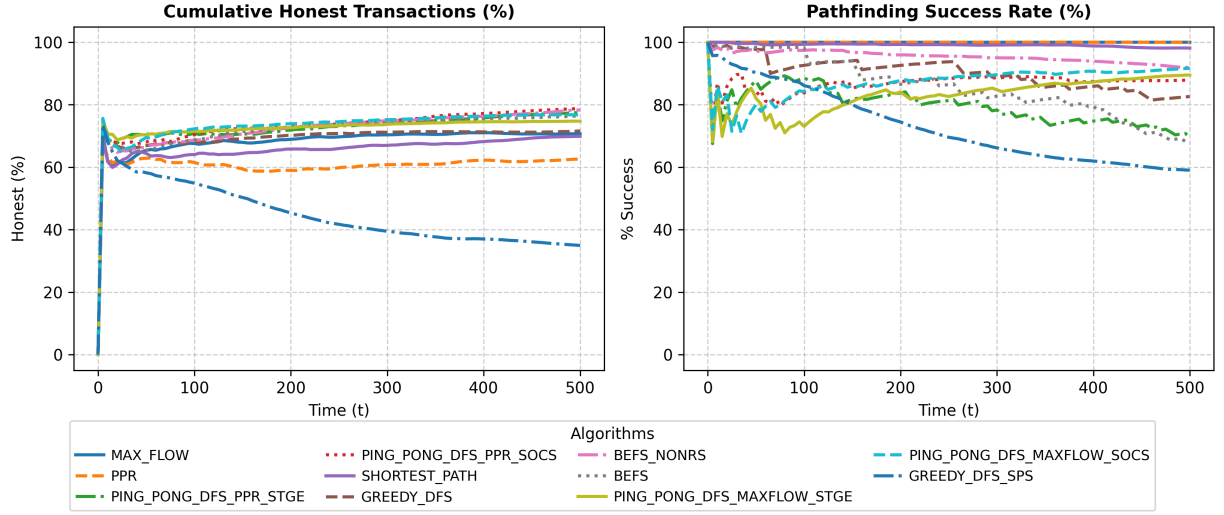


Figure 4.11: Performance of routing algorithms under Slander attack with candidate pool size = 5 ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

### Simulation Analysis - Malicious Rate: 0.51

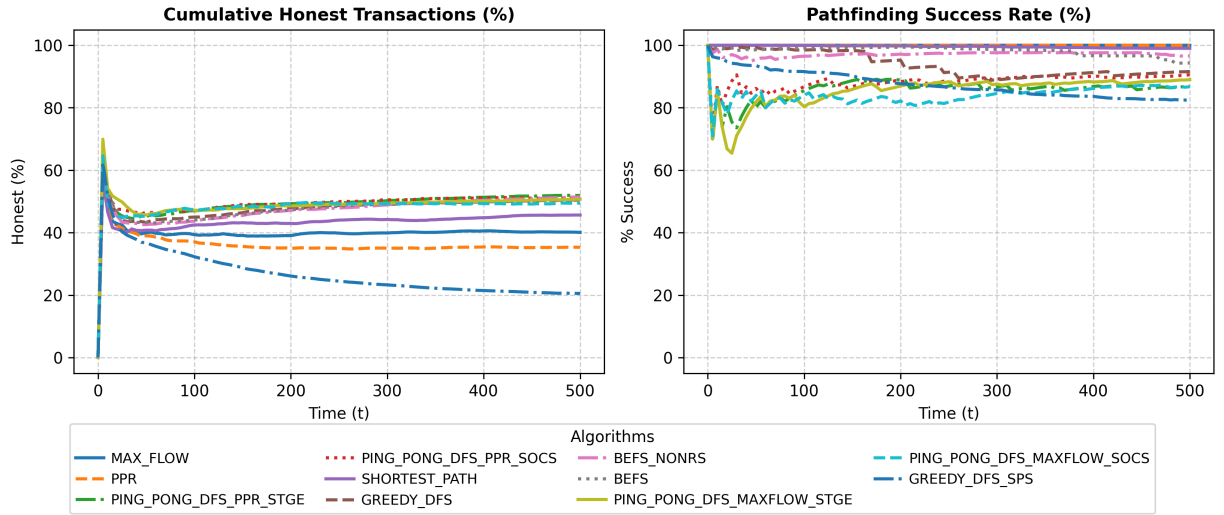


Figure 4.12: Performance of routing algorithms under Slander attack with candidate pool size = 5 ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

positive transactions is observable across all evaluated malicious rates ( $\alpha$ ). This positive trend applies to almost all routing algorithms, with the exception of **GREEDY\_DFS\_SPS** and **PPR**, which maintain the exact same behavior observed in the baseline slander scenario (Section 4.3.2).

However, this positive trend introduces notable side effects. Because candidate pooling selects the node with the highest reputation among 5 randomly sampled candidates, persistent slander depresses honest node metrics through continuous zero-value ratings. If malicious nodes execute scams infrequently according to their probability distribution, honest peers may eventually accumulate lower reputation scores than malicious ones, resulting in the starvation of honest destination nodes. This dynamic depends heavily on the malicious rate ( $\alpha$ ). At  $\alpha = 0.20$ , the probability of sampling a malicious candidate is  $1/5$ , making sustained slander difficult to maintain. Conversely, at  $\alpha = 0.51$ , the sampling probability rises to approximately  $1/2$ , significantly degrading long-term network performance.

In terms of the Pathfinding Success Rate  $P_{\text{success}}$ , performance remains largely unchanged, showing minor reductions at lower malicious rates while returning to baseline levels at higher adversarial ratios. Importantly, **GREEDY\_DFS\_SPS** bridges the performance gap observed in earlier simulations, achieving pathfinding stability comparable to all other algorithms. This could suggest that candidate pooling might mitigate vulnerabilities in fragile routing mechanisms under slander attacks.

Overall, candidate pooling proves to be an effective countermeasure against this attack scenario at low malicious rates. Its primary strength lies in its non-invasive nature, which mitigates adversarial impact without relying on aggressive penalization mechanisms, thereby enhancing network safety while avoiding false positives that could inadvertently isolate honest nodes. Conversely, significant security risks emerge as the proportion of malicious actors increases. Based on the current empirical results, it remains inconclusive whether candidate pooling alone provides sufficient protection under high malicious rates. Further evaluation with additional scenarios will be required to draw a definitive conclusion regarding its efficacy in high-density threat environments.

#### 4.3.5.3 Collusion Attack Scenario

This section evaluates candidate pooling under the collusion attack scenario. Figures 4.13 and 4.14 illustrate performance for a candidate pool size of 5 across malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$ , respectively.

Regarding the Cumulative Honest Transaction Rate  $T_{\text{honest}}$ , algorithm performance diverges into two distinct patterns compared to the baseline collusion scenario (Section 4.3.3).

On one hand, **GREEDY\_DFS**, **BEFS**, **BEFS\_NONRS**, and **PPR** exhibit a marked increase in  $T_{\text{honest}}$ , rising significantly above the baseline collusion threshold. This trend reflects the success

### Simulation Analysis - Malicious Rate: 0.2

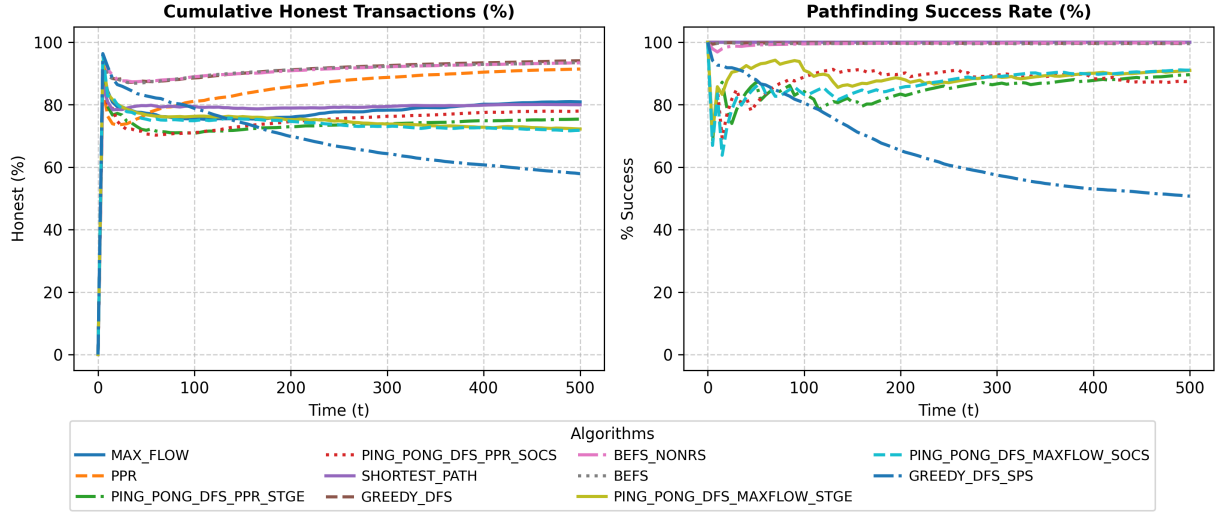


Figure 4.13: Performance of routing algorithms under Collusion attack with candidate pool size = 5 ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

### Simulation Analysis - Malicious Rate: 0.51

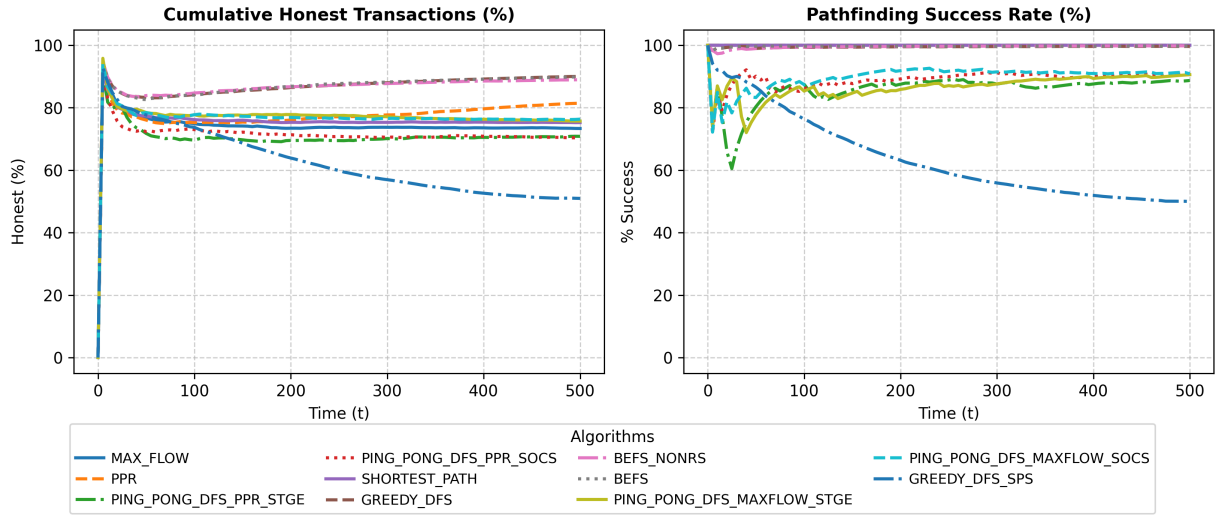


Figure 4.14: Performance of routing algorithms under Collusion attack with candidate pool size = 5 ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

of candidate pooling as a defense mechanism: by sampling 5 random candidates, these algorithms effectively bypass collusive trust rings. Consequently, transactions are routed to genuinely honest destination nodes rather than adversarial traps, successfully mitigating the attack.

On the other hand, ping-pong variants and the remaining routing algorithms show a reduction of roughly 5% in  $T_{\text{honest}}$ , falling below the baseline threshold. For these algorithms, candidate pooling becomes more of a disadvantage. Because collusion inflates their ratings, candidate pooling brings more malicious nodes into the top choices, leading to more honest nodes getting scammed. However, from a network monitoring perspective, this drop creates an anomaly that could trigger threshold-based intrusion detection systems.

In terms of the Pathfinding Success Rate  $P_{\text{success}}$ , values increase across all algorithms and malicious rates. At low malicious rates, this increase is helpful because candidate pooling offers more path options, increasing the chance of reaching honest nodes when there are few malicious nodes. However, as the malicious rate increases, this becomes less useful. When many malicious nodes are present and share high ratings, they can easily dominate the top candidate choices. Consequently, finding paths more easily is not very helpful, as almost any available path could be compromised, leading to a potential scam.

Overall, candidate pooling serves as an effective and non invasive defense under collusion attacks when paired with `GREEDY_DFS`, `BEFS`, `BEFS_NONRS`, or `PPR`. These algorithms achieve above average honest transaction rates  $T_{\text{honest}}$ , successfully avoiding the security risks connected to misleading Pathfinding Success Rates  $P_{\text{success}}$ . In contrast, the performance drop observed in the remaining algorithms could be used to trigger alarm systems, although using them means accepting that they bring insecurity and instability to the network.

#### 4.3.5.4 Slander-Collusion Attack Scenario

This section evaluates candidate pooling under the combined slander-collusion attack scenario. Figures 4.15 and 4.16 present the performance metrics for a pool size of 5 across malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$ , respectively.

Regarding the Cumulative Honest Transaction Rate  $T_{\text{honest}}$ , evaluating performance alone is difficult without metrics that track transaction volume among malicious nodes, as combining slander and collusion creates complex dynamics hard to read from a single plot. Nevertheless, clearer conclusions emerge when comparing these results with candidate pooling under collusion (Section 4.3.5.3), slander (Section 4.3.5.2), and the baseline slander-collusion scenario without pooling (Section 4.3.4).

An interesting observation across all analyzed plots is that candidate pooling favors a positive shift in cumulative honest transactions. However, as previously analyzed, this remains a positive outcome only as long as the malicious rate is low: with a source node

### Simulation Analysis - Malicious Rate: 0.2

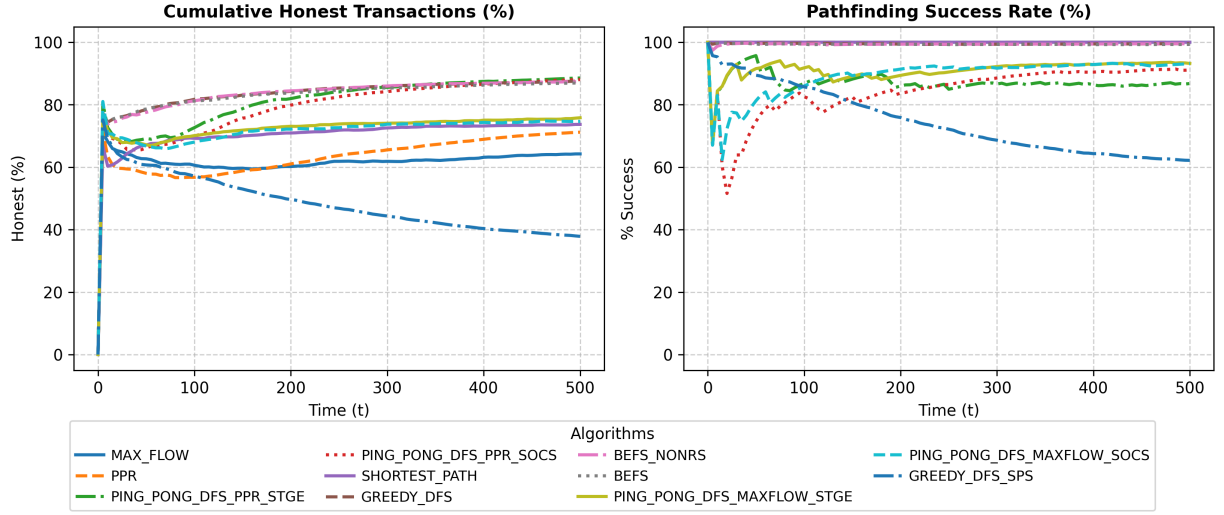


Figure 4.15: Performance of routing algorithms under Slander-Collusion attack with candidate pool size = 5 ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

### Simulation Analysis - Malicious Rate: 0.51

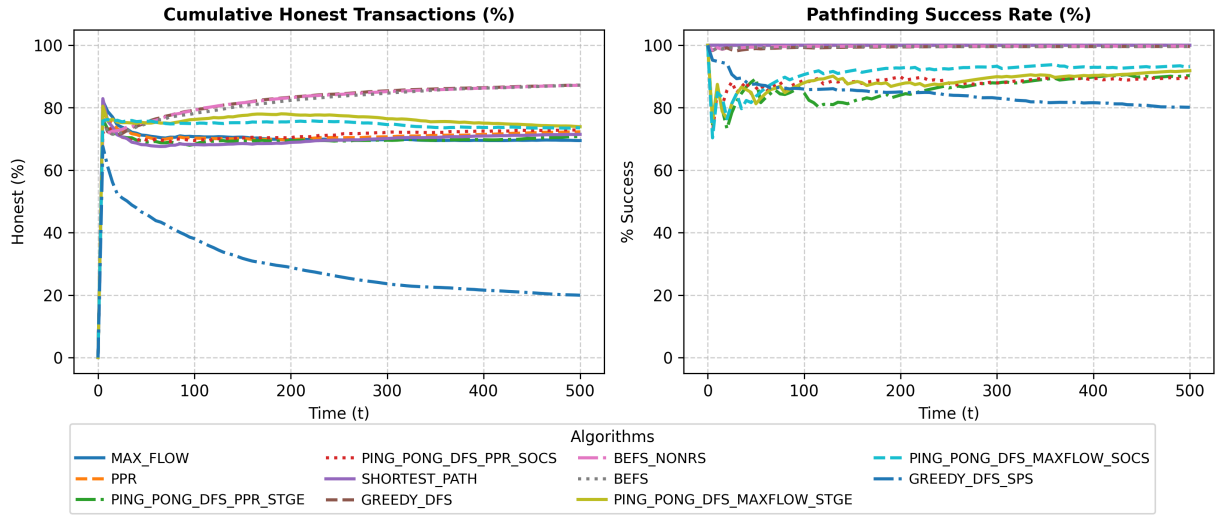


Figure 4.16: Performance of routing algorithms under Slander-Collusion attack with candidate pool size = 5 ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$  and  $P_{\text{success}}$  over time.

probability of being malicious at  $1/5$  and a destination node probability at  $1/5$ , probability alone suggests these transactions are genuinely honest rather than inflated by collusion. The situation changes at a malicious rate of  $\alpha = 0.51$ , where these probabilities rise to  $1/2$  for both source and destination nodes, preventing a definitive interpretation for average results. The only exception lies in a few algorithms that push their values significantly higher than the rest, suggesting that whenever a candidate pool is formed, the top choice is genuinely an honest node that receives a positive evaluation, effectively defending against this attack vector.

Once again, `GREEDY_DFS`, `BEFS` and `BEFS_NONRS` reach peaks above 80% across all malicious rates, demonstrating their capability to manage high-severity combined attacks. Trailing slightly behind with values just under 80%, `PINGPONG_DFS_PPR_STGE` and `PINGPONG_DFS_PPR_SOCS` achieve a 20% increase in positive transactions compared to the scenario without candidate pooling. For the remaining algorithms, performance remains virtually identical, leading to the conclusion that even a minor increase does not represent a real mitigation: when half the network is malicious and executing positive transactions among themselves with a 50% chance of being the source, candidate pooling under these conditions cannot be considered reliable.

Regarding the Pathfinding Success Rate  $P_{\text{success}}$ , values increase across all algorithms and across all malicious rates. Logically, this positive effect stems from the fact that leveraging a candidate pool provides more choices: if a path to one node cannot be found, alternative candidates offer other potential paths. Furthermore, as observed at higher malicious rates, these values increase even more due to collusion, meaning many of the additional paths found pass directly through malicious nodes. Consequently, this metric requires careful interpretation from a security perspective. Once again, the primary beneficiaries are `GREEDY_DFS`, `BEFS` and `BEFS_NONRS`, which manage to increase their successful paths even further while serving a high volume of genuinely honest transactions.

Overall, evaluating both metrics demonstrates that candidate pooling provides tangible benefits across routing metrics under low malicious rates. However, at high malicious rates ( $\alpha = 0.51$ ), these advantages largely deteriorate, leaving `GREEDY_DFS`, `BEFS`, `BEFS_NONRS`, `PINGPONG_DFS_PPR_STGE`, and `PINGPONG_DFS_PPR_SOCS` as the only algorithms capable of sustaining performance, even though paths formed under this dual attack vector carry inherent security risks. For the remaining algorithms, the risk of transaction starvation for honest nodes persists, driven by candidate pool selection dynamics, collusive rating inflation, and distrust induced by slander. Despite these initial analyses, greater certainty and a deeper examination of these dynamics will be provided in Section 4.3.9.1.

### 4.3.6 Single punishment Defense

To evaluate the single punishment mechanism described in Section 3.4.2, we re-examine all attack scenarios previously analyzed in Section 4.3.5, with the sole addition of the single punishment option (`--punishment single`) to isolate its specific impact on network performance. This preliminary setup provides initial insights that serve as a foundation for the comprehensive transaction analysis conducted in Section 4.3.9.2.

#### 4.3.6.1 Normal Run Scenario

This section begins with a baseline evaluation under normal operational conditions. Figures 4.17 and 4.18 present all three performance metrics across malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$  under the single punishment mechanism.

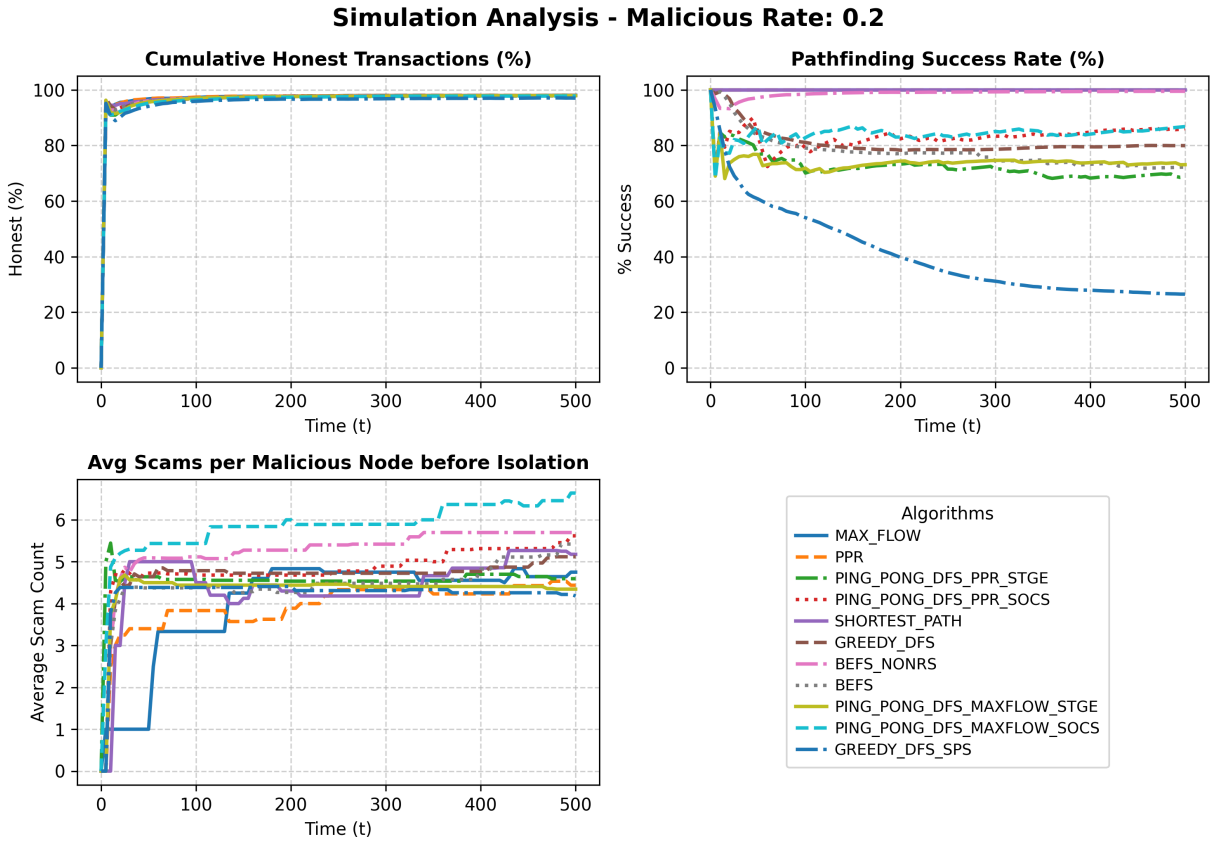


Figure 4.17: Performance of routing algorithms under Normal Run with single punishment defense ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

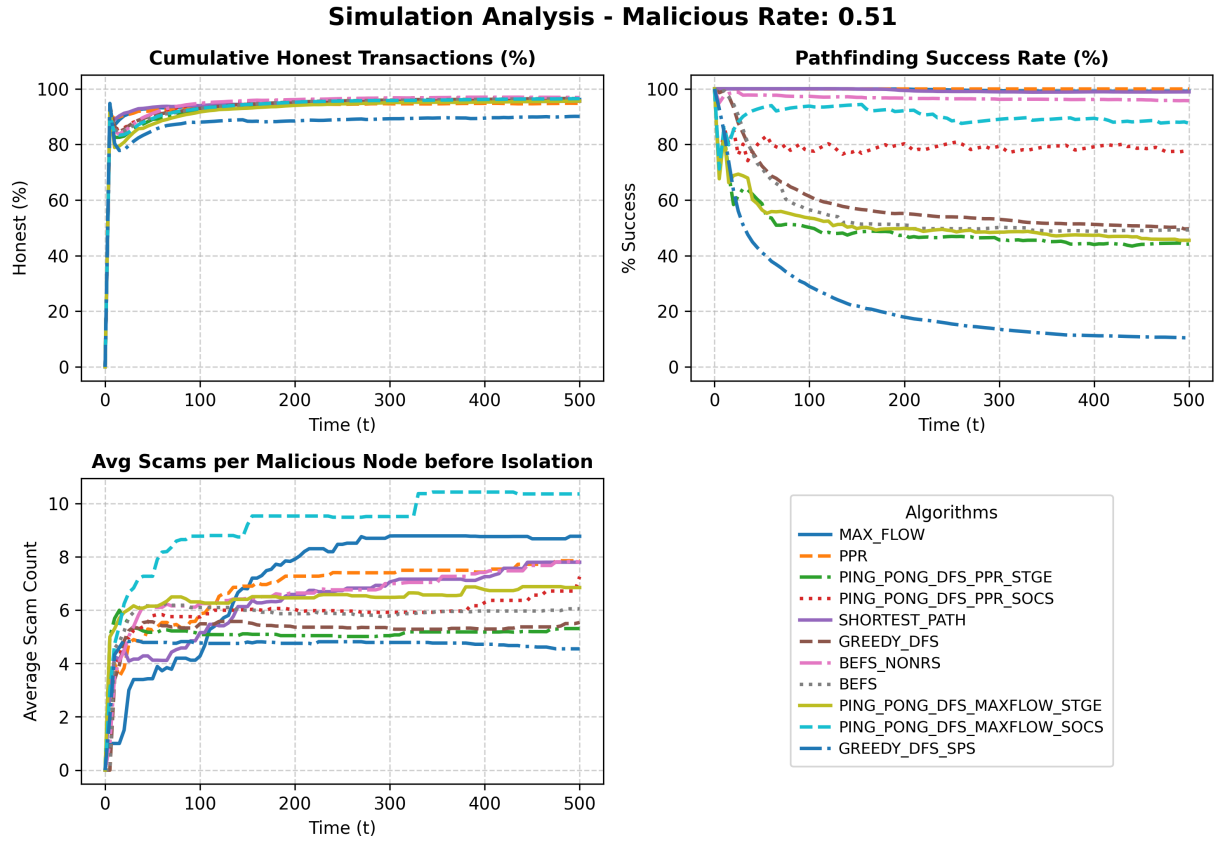


Figure 4.18: Performance of routing algorithms under Normal Run with single punishment defense ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

Regarding the Cumulative Honest Transaction Rate  $T_{\text{honest}}$ , the single punishment mechanism yields a substantial performance improvement across all malicious rates compared to candidate pooling under normal operation (Section 4.3.5.1). In particular, at  $\alpha = 0.51$ , nearly all algorithms converge above 90%, with `GREEDY_DFS_SPS` being the sole exception, remaining slightly below this threshold.

However, the main drawback of this aggressive defense is its negative impact on the Pathfinding Success Rate  $P_{\text{success}}$ . This degradation is already visible at a low malicious rate ( $\alpha = 0.20$ ), where pathfinding decreases compared to the candidate pooling scenario, mainly driven by a sharp drop in `GREEDY_DFS_SPS`. As the malicious rate rises to  $\alpha = 0.51$ , this negative trend expands and splits the algorithms into two distinct groups.

The first group includes `PINGPONG_DFS_PPR_STGE`, `PINGPONG_DFS_MAXFLOW_STGE`, `GREEDY_DFS_SPS`, and notably `GREEDY_DFS` and `BEFS`, which all drop below a 60% success rate. This sharp decline shows that severe penalties cause network fragmentation, trading path availability for cleaner transactions. In contrast, the remaining algorithms handle this aggressive defense well and keep their success rates above 80%, staying consistent with earlier baselines.

Finally, the average number of scams  $S_{\text{avg}}$  shows that this aggressive defense isolates malicious nodes rapidly, particularly at high malicious rates ( $\alpha = 0.51$ ). A clear correlation emerges between eviction speed and pathfinding performance. Specifically, the algorithms that isolate malicious nodes most rapidly, capping at roughly 6 average scams before stopping the threat, are precisely those that suffer the lowest Pathfinding Success Rates. In contrast, algorithms that take more time to identify and isolate malicious nodes, such as `PINGPONG_DFS_MAXFLOW_SOCS`, `PINGPONG_DFS_PPR_SOCS`, `MAXFLOW`, `PPR`, `SHORTEST_PATH`, and `BEFS_NONRS`, maintain significantly higher Pathfinding Success Rates.

This trade-off between rapid threat isolation and network connectivity is especially critical in small-scale networks like the simulated topology (100 to 150 nodes), where node eviction directly reduces path availability. In conclusion, under normal operational conditions with an invasive defense mechanism, `BEFS_NONRS` proves to be the best-performing algorithm overall. Conversely, standard `BEFS` and `GREEDY_DFS` fail to withstand the impact of aggressive penalties. Among embedding-based approaches, `PINGPONG_DFS_MAXFLOW_SOCS` and `PINGPONG_DFS_PPR_SOCS` emerge as the most effective and resilient alternatives in this context.

#### 4.3.6.2 Slander Attack Scenario

Building upon our earlier evaluations, we now analyze system resilience under a Slander attack managed by the single punishment defense mechanism. Figures 4.19 and 4.20 present the evaluation metrics ( $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$ ) across malicious rates of  $\alpha = 0.20$  and

$\alpha = 0.51$ .

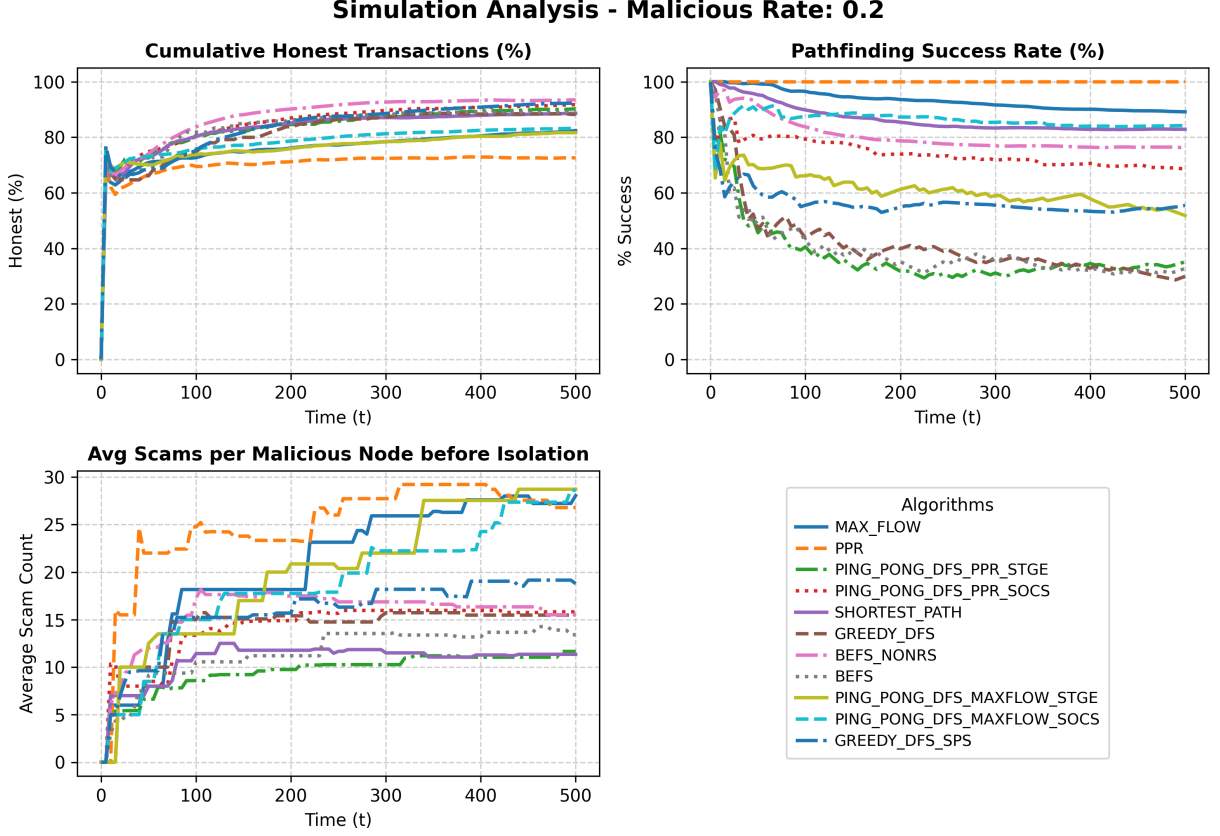


Figure 4.19: Performance of routing algorithms under Slander attack with single punishment defense ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

Evaluating the single punishment defense under Slander attacks reveals severe performance trade-offs when compared to candidate pooling (Section 4.3.5.2). At a low malicious rate ( $\alpha = 0.20$ ), Pathfinding Success Rates  $P_{\text{success}}$  drop sharply across nearly all algorithms, despite a minor increase in  $T_{\text{honest}}$ . This pathfinding collapse is particularly alarming because such degradation was not observed during the pooling run (Section 4.3.5.2). This provides an initial indication that the defense mechanism suffers indirectly from the attack, as false evaluations induce network fragmentation. Furthermore, the average number of scams  $S_{\text{avg}}$  required to isolate malicious nodes increases noticeably across all algorithms. Under these conditions, the only embedding-based approaches that preserve overall resilience for this malicious rate are BEFS\_NONRS, PINGPONG\_DFS\_PPR\_SOCS, and PINGPONG\_DFS\_MAXFLOW\_SOCS.

Conversely, at  $\alpha = 0.51$ , the combined impact of slander attacks and aggressive punishment severely destabilizes the network. Pathfinding performance degrades significantly even for

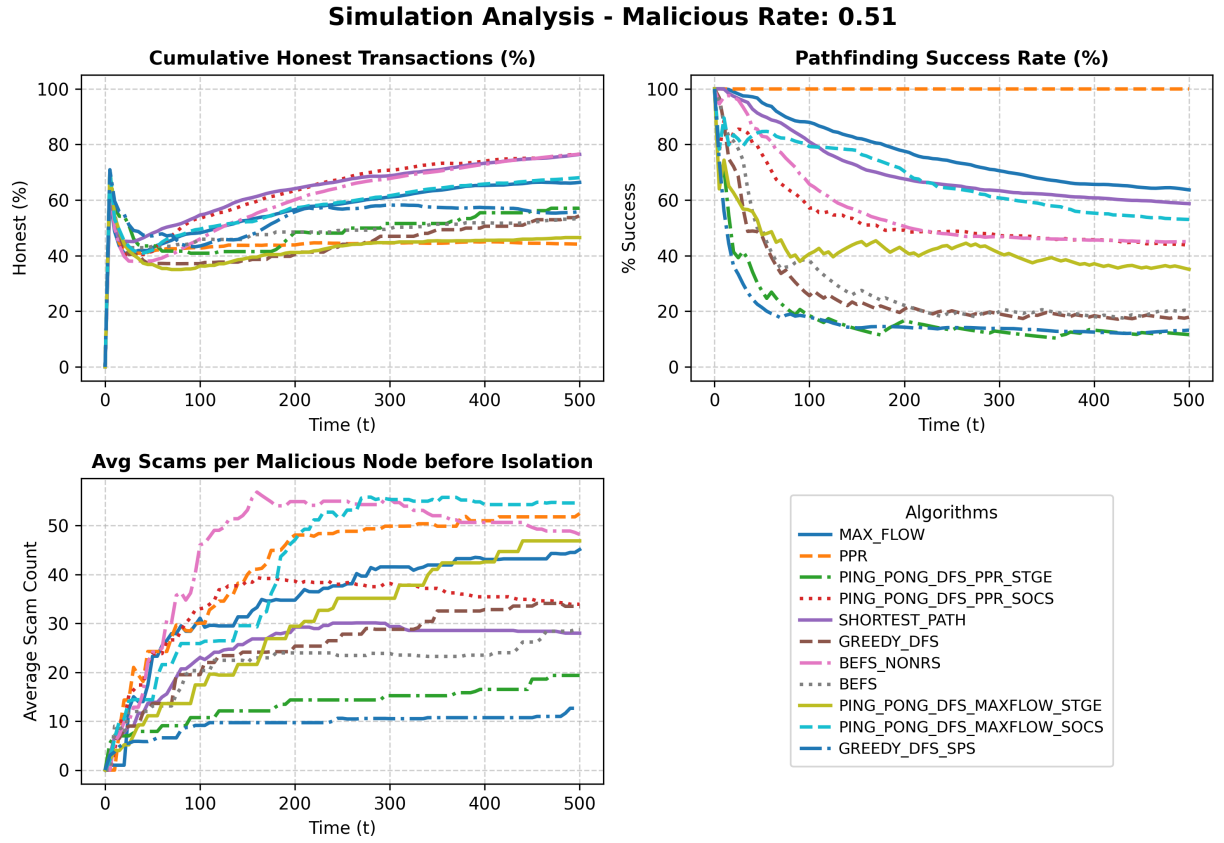


Figure 4.20: Performance of routing algorithms under Slander attack with single punishment defense ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

robust global baselines like **MAXFLOW** and **SHORTEST\_PATH**, which fall to roughly 60%  $P_{\text{success}}$ . Top embedding approaches also experience severe drops, with **PINGPONG\_DFS\_MAXFLOW\_SOCS** reaching 57%, while **BEFS\_NONRS** and **PINGPONG\_DFS\_PPR\_SOCS** drop slightly below 50%. In practice, under high attack intensity, nearly one out of two pathfinding attempts fails. Most remaining algorithms drop to around 20% success rate, with **PINGPONG\_DFS\_MAXFLOW\_STGE** reaching 37%. However, when paths are successfully found, **BEFS\_NONRS** and **PINGPONG\_DFS\_PPR\_SOCS** maintain high transaction quality with an 80% honest transaction rate  $T_{\text{honest}}$ , whereas **PINGPONG\_DFS\_MAXFLOW\_SOCS** achieves roughly 65%, and all other algorithms stay close to the values recorded under candidate pooling (Section 4.3.5.2).

The main weakness of this configuration lies in threat isolation  $S_{\text{avg}}$ . Under high adversarial presence ( $\alpha = 0.51$ ), even the best-performing algorithms require between 35 scams for **PINGPONG\_DFS\_PPR\_SOCS** and up to 50 scams for **PINGPONG\_DFS\_MAXFLOW\_SOCS** and **BEFS\_NONRS** before isolating adversaries. This indicates that despite aggressive penalties, slander attacks allow malicious nodes to persist and execute repeated attacks. This instability occurs because false slander warnings cause the defense mechanism to indirectly penalize honest nodes, disrupting network trust and pathfinding. Ultimately, only **BEFS\_NONRS**, **PINGPONG\_DFS\_MAXFLOW\_SOCS**, and **PINGPONG\_DFS\_PPR\_SOCS** maintain acceptable performance across both attack rates, whereas all other algorithms fail. Adopting this defense thus involves a clear trade-off: transaction quality remains high and pathfinding stays acceptable, but at the cost of delayed threat eviction.

#### 4.3.6.3 Collusion Attack Scenario

Building upon our earlier evaluations, we now analyze system resilience under a Collusion attack managed by the single punishment defense mechanism. Figures 4.21 and 4.22 present the evaluation metrics ( $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$ ) across malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$ .

Evaluating the single punishment defense against candidate pooling under collusion (Section 4.3.5.3) reveals a significant increase in Cumulative Honest Transaction Rate  $T_{\text{honest}}$  across all malicious rates ( $\alpha$ ). This improvement occurs because, unlike slander attacks, which directly target honest node evaluations and induce widespread confusion through continuous unwarranted penalties, collusion relies on reciprocal boosting among malicious peers. Under this boosting dynamic, detection occurs much faster: honest nodes retain high reputation scores, enabling the defense mechanism to execute punishments far more decisively and effectively once a scam occurs, rather than indirectly penalizing honest peers as observed under slander.

In parallel, pathfinding success  $P_{\text{success}}$  remains largely preserved, experiencing only minor degradation (capped between 5% and 10%) relative to candidate pooling across nearly all embedding schemes, with the sole exception of **GREEDY\_DFS\_SPS**, which collapses as

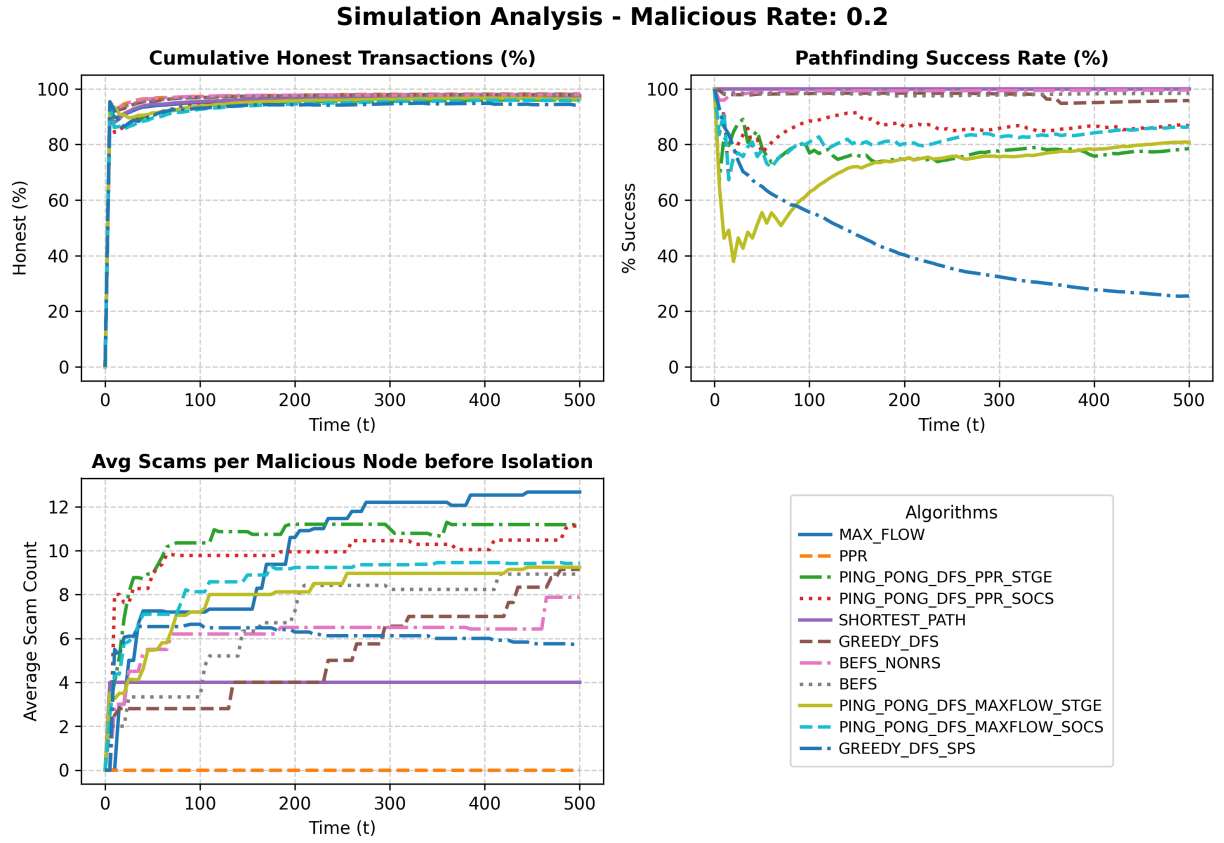


Figure 4.21: Performance of routing algorithms under Collusion attack with single punishment defense ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

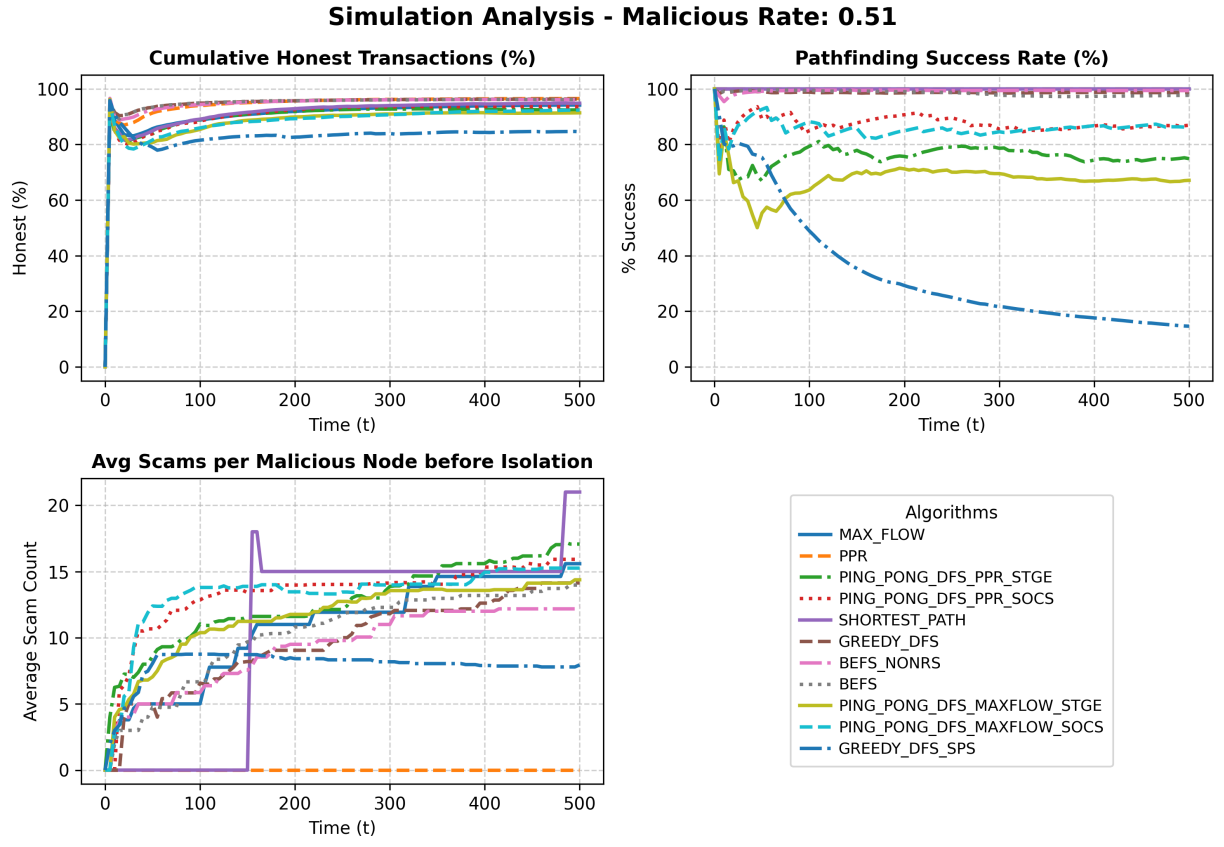


Figure 4.22: Performance of routing algorithms under Collusion attack with single punishment defense ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

in previous evaluations. This slight reduction represents an acceptable trade-off given the substantial gains in transaction quality and overall network health. Regarding threat isolation  $S_{\text{avg}}$ , the defense maintains manageable levels, requiring between 8 and 11 scams for embedding algorithms at  $\alpha = 0.20$ , and between 12 and 16 scams at  $\alpha = 0.51$ . However, this drawback remains marginal because honest transaction throughput stays remarkably high, effectively rendering malicious nodes harmless despite persisting longer within the network.

When evaluating overall trade-offs across  $P_{\text{success}}$ ,  $S_{\text{avg}}$ , and  $T_{\text{honest}}$ , **GREEDY\_DFS**, **BEFS\_NONRS** and **BEFS** consistently rank among the top performers. These schemes are closely followed by **PINGPONG\_DFS\_MAXFLOW\_SOCS** and **PINGPONG\_DFS\_PPR\_SOCS**, whereas the remaining algorithms trail behind, leaving **GREEDY\_DFS\_SPS** as the sole scheme failing under adversarial conditions. Overall, pairing single punishment with top candidate selection forms an effective safeguard against collusive behavior, maintaining network health and security. Even if a few malicious nodes persist, in most cases they are isolated from being selected as destination nodes through top candidate selection, preventing further scams and proving that single punishment successfully neutralizes reciprocal boosting dynamics.

#### 4.3.6.4 Slander-Collusion Attack Scenario

Building upon our previous evaluations, we finally analyze system resilience under a combined slander-collusion attack managed by the single punishment defense mechanism. Figures 4.23 and 4.24 present the evaluation metrics ( $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$ ) across malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$ .

Evaluating the single punishment defense under combined slander-collusion attacks reveals a complex and counterintuitive behavioral pattern. On one hand, cumulative honest transactions  $T_{\text{honest}}$  increase significantly compared to the baseline candidate pooling scenario (Section 4.3.5.4) across all malicious rates ( $\alpha$ ), despite a general reduction in pathfinding success  $P_{\text{success}}$ . However, the primary anomaly lies in threat isolation  $S_{\text{avg}}$ , which remains strikingly static and fails to effectively evict malicious nodes across all malicious rates.

This stagnation in threat isolation likely comes from how the combined attack directly subverts the defense mechanism. By pairing slander with reciprocal boosting, malicious actors exploit the protocol rules, preventing the eviction of malicious nodes and effectively turning the single punishment mechanism against honest ones. Under this scenario, a potential explanation for the high cumulative honest transactions  $T_{\text{honest}}$  is that slandering honest nodes while boosting colluding peers severely distorts network topology. As a result, honest nodes are repeatedly penalized and progressively isolated from the system, leaving malicious actors to form a larger portion of the active network.

If this premise holds, the transaction flow shifts drastically: malicious-to-malicious in-

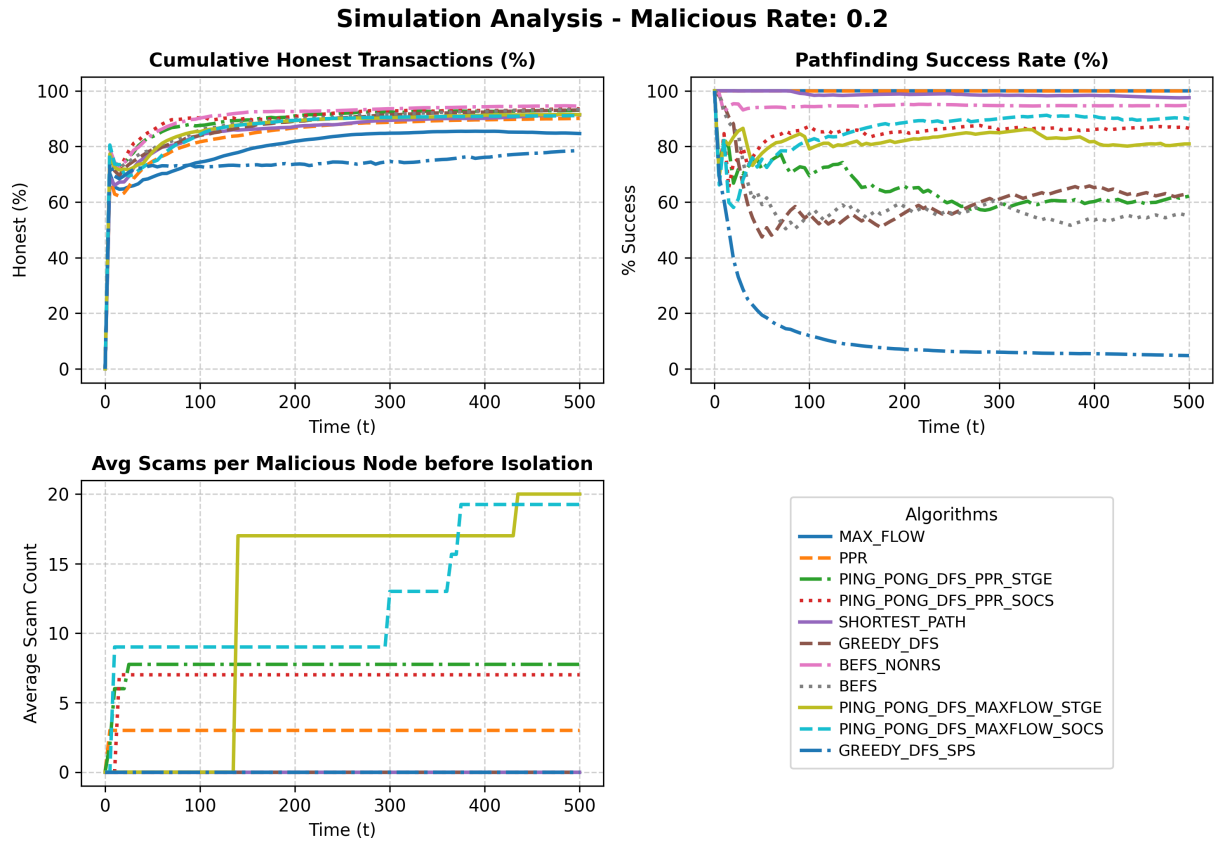


Figure 4.23: Performance of routing algorithms under slander-collusion attack with single punishment defense ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

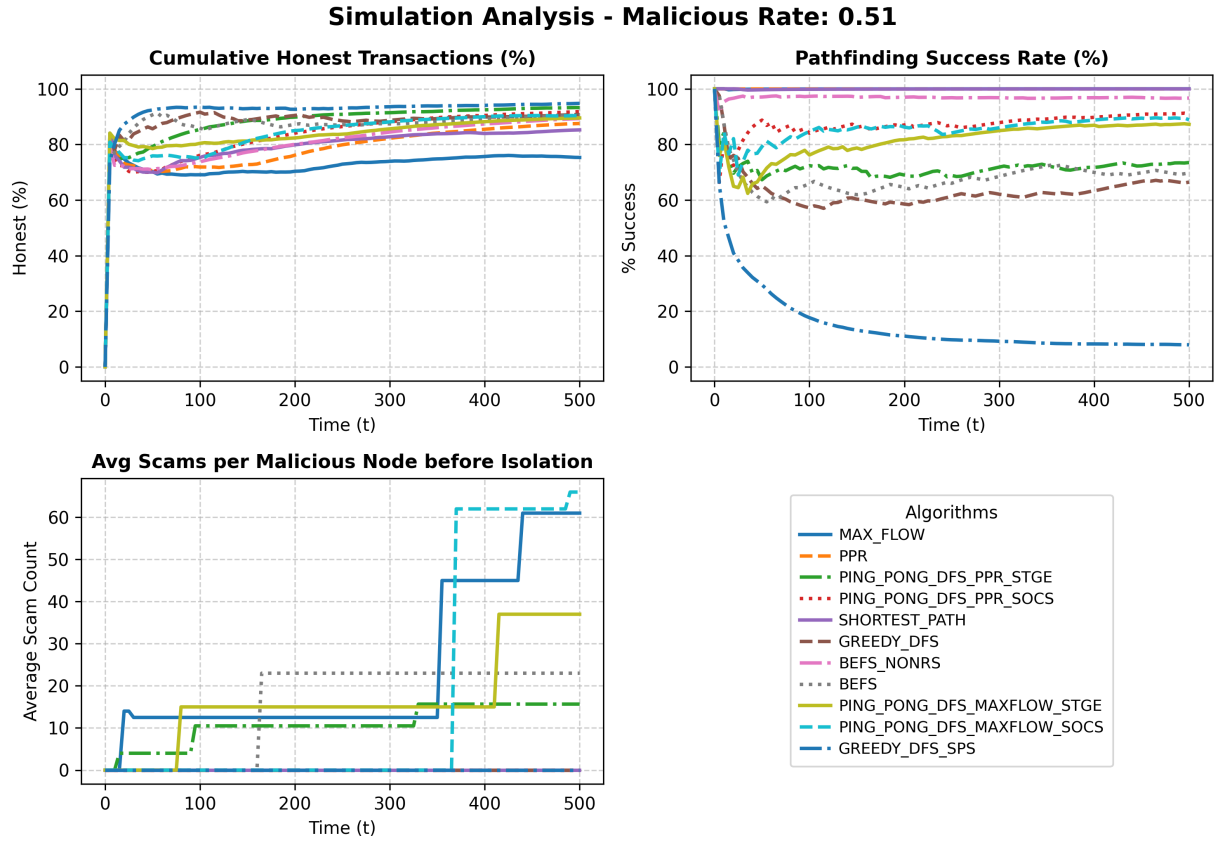


Figure 4.24: Performance of routing algorithms under slander-collusion attack with single punishment defense ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

teractions would represent the vast majority, followed by a smaller fraction of honest-to-malicious attempts. Conversely, honest-to-honest and malicious-to-honest transactions would drop to near zero because slandered honest nodes never reach top candidate selection due to reputation inflation. This explanation is further supported by comparing these findings with the pure slander scenario under single punishment (Section 4.3.6.2), where transaction counts show an unnatural surge, particularly at  $\alpha = 0.51$ .

To test this structural hypothesis, Section 4.3.9.2 will evaluate the directional transaction flow by specifically measuring how many honest nodes initiate transactions versus how many are selected as destination nodes. Until these interaction dynamics are fully analyzed, a definitive verdict regarding the true efficacy of the single punishment defense against slander-collusion attacks remains premature.

### 4.3.7 Recommender Punishment Defense

To evaluate the recommender punishment mechanism described in Section 3.4.4, we do not test all the attacks analyzed in previous sections. For most attacks, the defense shows no real impact because it fails to isolate malicious nodes from the network. Additionally, as explained in Section 3.4.4, this defense only applies to the shortest path routing metric, as it is the only one that returns a specific path to operate on. All simulation settings remain identical to the trade pooling setup in Section 4.3.5, with the sole addition of the recommender punishment option (`--punishment recommender`) to test its impact on network performance.

#### 4.3.7.1 Slander and Slander-Collusion Attack Scenarios

Building upon our earlier evaluations, we now analyze system resilience under Slander and Slander-Collusion attacks managed by the recommender punishment defense mechanism. Figures 4.25 and 4.26 present the evaluation metrics ( $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$ ) under the Slander attack across malicious rates of  $\alpha = 0.20$  and  $\alpha = 0.51$ .

In particular, the Slander attack represents the only scenario where the recommender punishment defense successfully removes malicious nodes from the system. In these results, the number of evicted malicious nodes increases with higher malicious rates, even though  $S_{\text{avg}}$  remains relatively high.

Compared to the trade pooling setup in Section 4.3.5.2,  $T_{\text{honest}}$  increases for both malicious rates while pathfinding performance declines. This shows that the mechanism works effectively when attackers only target honest nodes. However, the defense depends entirely on the absence of collusion among malicious entities.

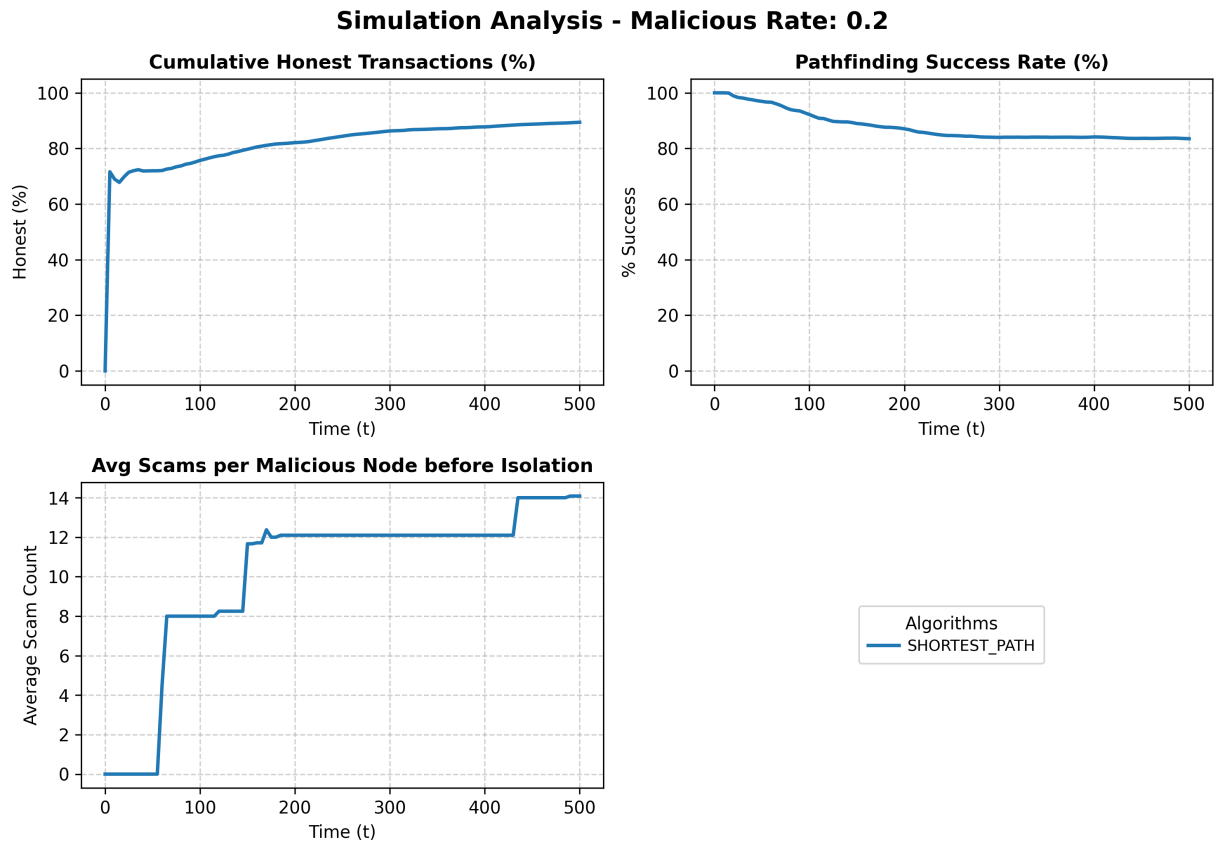


Figure 4.25: Performance of routing algorithms under Slander attack with recommender punishment defense ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

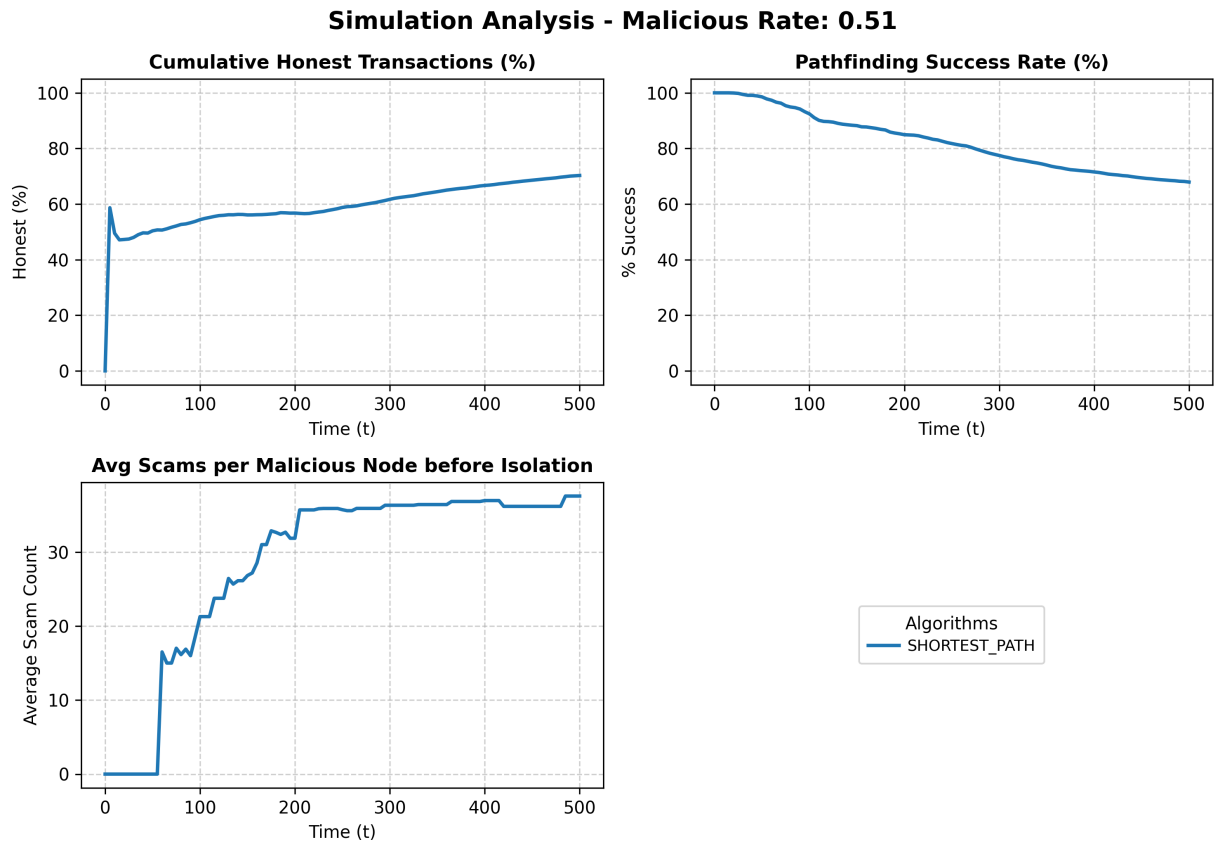


Figure 4.26: Performance of routing algorithms under Slander attack with recommender punishment defense ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

This vulnerability becomes clear in the Slander-Collusion scenario, shown in Figures 4.27 and 4.28. When attackers combine slander with mutual reputation boosting, evictions drop strictly back to zero, causing the defense mechanism to fail in evicting malicious nodes despite an increase in  $T_{\text{honest}}$  of 15% for  $\alpha = 0.51$  and 20% for  $\alpha = 0.20$  compared to Section 4.3.5.4.

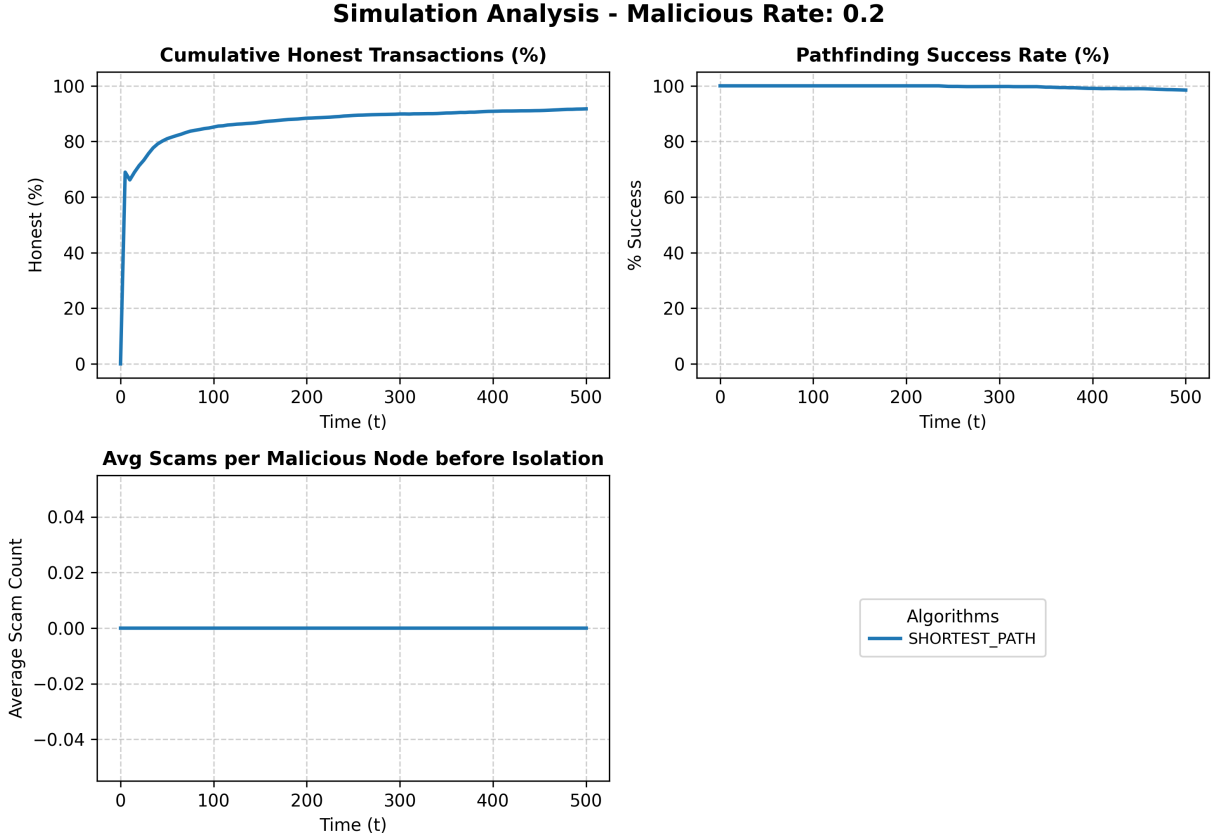


Figure 4.27: Performance of routing algorithms under Slander-Collusion attack with recommender punishment defense ( $\alpha = 0.20$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

Consequently, while the recommender punishment mechanism partially mitigates isolated Slander attacks, the introduction of mutual boosting causes the defense to fail completely in evicting malicious nodes, allowing them to operate freely within the system. Although honest transactions continue to increase, this severe limitation leaves the network vulnerable, as combining it with more aggressive attack vectors could cause far greater damage. Because driving evictions entirely to zero is a critical failure, this defense is currently considered unsuitable for deployment.

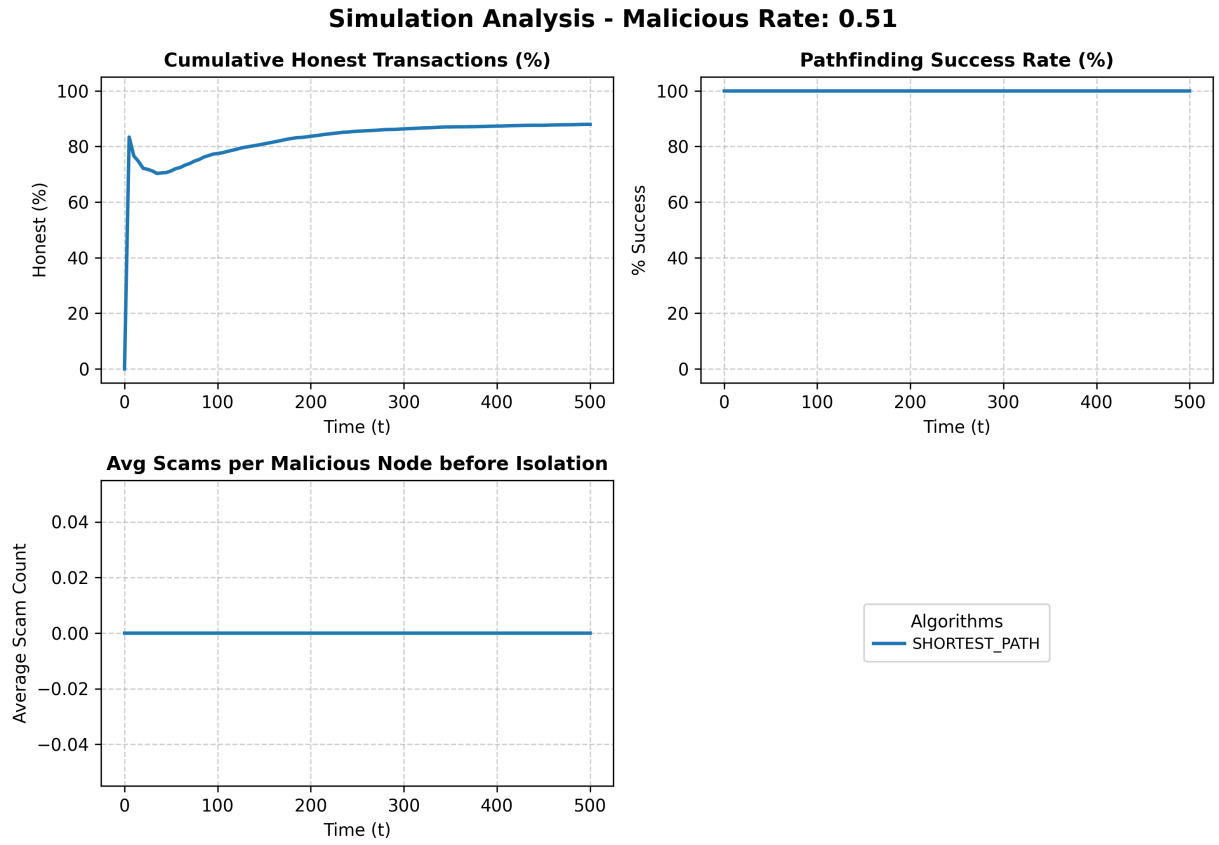


Figure 4.28: Performance of routing algorithms under Slander-Collusion attack with recommender punishment defense ( $\alpha = 0.51$ ), showing  $T_{\text{honest}}$ ,  $P_{\text{success}}$ , and  $S_{\text{avg}}$  over time.

### 4.3.8 Path Punishment Defense

With the path punishment defense, overall network behavior stays almost the same as in the trade pooling setup (Section 4.3.5), showing only small improvements in Cumulative Honest Transaction Rate  $T_{\text{honest}}$ . However, this mechanism fails to isolate or evict any malicious nodes across all attack scenarios and malicious rates ( $\alpha$ ). Since the resulting plots show no new insights and look nearly identical to previous baselines, we omit them from this section.

This happens because the penalties tend to be too weak, as the main goal of this defense was to be less invasive. However, unlike the recommender punishment defense (Section 4.3.7), overall performance does not increase significantly here. Combined with the fact that it fails to isolate malicious nodes, this defense is currently ineffective.

Future work could focus on finding a better value for the numerator in Equation 3.11 to increase the strength of the punishment, though this would defeat its main goal of being a non-invasive technique.

### 4.3.9 Transaction Analysis of Honest and Malicious Nodes

To gain a deeper understanding of how our defenses work, we analyze the nodes that initiate transactions and those selected to receive them. This allows us to address the questions raised in the previous slander collusion analysis (Section 4.3.6.4) while providing a clear general picture. In addition, we examine a final metric that tracks how many times honest and malicious nodes are cheated.

Unlike previous analyses that showed performance over time, these data refer specifically to the end of each simulation. As in previous sections, we focus on malicious rates ( $\alpha$ ) of 0.20 and 0.51 to avoid an unnecessary excess of data.

All percentages reported in the following tables are normalized relative to the total number of transactions executed in the network ( $T_{\text{total}}$ ). The metrics are structured into two categories:

- **Binary Split Metrics:** Metrics composed of two complementary parts that sum to 100%:

- $P_{\text{honest\_src}}$ : Calculated as  $P_{\text{honest\_src}} = \left( \frac{T_{\text{honest\_src}}}{T_{\text{total}}} \right) \times 100$ , where  $T_{\text{total}} = T_{\text{honest\_src}} + T_{\text{malicious\_src}}$ , measuring transactions initiated by honest source nodes.
- $P_{\text{honest\_dst}}$ : Calculated as  $P_{\text{honest\_dst}} = \left( \frac{T_{\text{honest\_dst}}}{T_{\text{total}}} \right) \times 100$ , where  $T_{\text{total}} = T_{\text{honest\_dst}} + T_{\text{malicious\_dst}}$ , measuring transactions received by honest destination nodes.

- **Trade Outcome Partitioning:** Metrics forming a 4-way partition that sums to the total transaction volume ( $T_{\text{total}}$ ):

$$T_{\text{total}} = T_{\text{honest\_good}} + T_{\text{honest\_bad}} + T_{\text{malicious\_good}} + T_{\text{malicious\_bad}} \quad (4.4)$$

- $P_{\text{honest\_bad}}$ : Calculated as  $P_{\text{honest\_bad}} = \left( \frac{T_{\text{honest\_bad}}}{T_{\text{total}}} \right) \times 100$ , measuring cheated transactions involving honest nodes out of all network trades  $T_{\text{total}}$ .
- $P_{\text{malicious\_bad}}$ : Calculated as  $P_{\text{malicious\_bad}} = \left( \frac{T_{\text{malicious\_bad}}}{T_{\text{total}}} \right) \times 100$ , measuring cheated transactions involving malicious nodes out of all network trades  $T_{\text{total}}$ .

To present these results, the evaluation is divided into two main sections: the first examines the impact of candidate pooling, while the second focuses on single node punishment.

#### 4.3.9.1 Trade Pooling Defense

We begin by evaluating the Trade Pooling defense mechanism. Table 4.3 reports the percentage of honest nodes initiating transactions, while Table 4.4 presents the percentage of honest nodes selected as destination targets to provide the service. Additionally, Table 4.5 shows the percentage of honest nodes that get cheated, and Table 4.6 reports the percentage of cheated malicious nodes.

The configuration used for this scenario is (`--punishment none`) and (`--poolTrade 5`).

Once again, cross-referencing these findings with our earlier results provides a clearer understanding of the underlying dynamics. As noted in the baseline scenario analysis (Section 4.3.5.1), all embedding metrics exhibited a significant increase in cumulative honest transactions. Examining destination selection in Table 4.4 explains this trend: destination targets are predominantly composed of honest nodes across almost all metrics, with the exceptions of `PINGPONG_DFS_MAXFLOW_STGE`, `PINGPONG_DFS_MAXFLOW_SOCS`, and the usual `GREEDY_DFS_SPS`.

Re-evaluating the cheated transaction metrics for honest and malicious nodes (Table 4.5 and Table 4.6) confirms that these same algorithms remain less effective at mitigating dishonest transactions, as malicious nodes appear more frequently within their candidate pools compared to other metrics. Nevertheless, despite these drawbacks, `PINGPONG_DFS_MAXFLOW_STGE` and `PINGPONG_DFS_MAXFLOW_SOCS` maintain respectable overall performance. Consequently, candidate pool selection under baseline conditions demonstrates high efficacy for nearly all embedding algorithms, with the sole exception of `GREEDY_DFS_SPS`, thereby confirming our previous observations.

Regarding the collusion scenario analysis (Section 4.3.5.3), the destination selection table (Table 4.4) once again confirms our previous theory. Specifically, `BEFS_NONRS`, `BEFS`, and

Table 4.3: Trade Pooling Defense: Percentage of honest nodes initiating transactions as source nodes at the end of the simulation.

| Malicious Rate ( $\alpha$ ) | Normal Run |      | Collusion |      | Slander |      | Slander-Collusion |      |
|-----------------------------|------------|------|-----------|------|---------|------|-------------------|------|
|                             | 0.20       | 0.51 | 0.20      | 0.51 | 0.20    | 0.51 | 0.20              | 0.51 |
| max_flow                    | 74.5       | 45.7 | 75.4      | 45.8 | 74.5    | 45.7 | 75.4              | 45.8 |
| ppr                         | 74.5       | 45.7 | 75.4      | 45.8 | 74.5    | 45.7 | 75.4              | 45.8 |
| ping_pong_dfs_ppr_stge      | 76.7       | 44.7 | 73.8      | 43.9 | 79.8    | 51.2 | 77.3              | 45.4 |
| ping_pong_dfs_ppr_socs      | 75.4       | 52.1 | 74.8      | 44.3 | 74.1    | 39.6 | 79.9              | 49.3 |
| shortest_path               | 74.5       | 45.7 | 75.4      | 45.8 | 74.9    | 45.5 | 75.4              | 45.8 |
| greedy_dfs                  | 75.3       | 47.3 | 78.4      | 49.5 | 74.8    | 47.9 | 76.4              | 47.0 |
| befs_nonrs                  | 74.5       | 49.6 | 75.9      | 44.1 | 82.1    | 49.4 | 75.2              | 47.5 |
| befs                        | 75.7       | 47.7 | 75.2      | 45.4 | 80.2    | 47.7 | 76.7              | 48.7 |
| ping_pong_dfs_maxflow_stge  | 75.1       | 47.3 | 76.0      | 46.9 | 75.8    | 49.6 | 75.7              | 46.3 |
| ping_pong_dfs_maxflow_socs  | 76.0       | 50.2 | 75.7      | 44.2 | 74.5    | 45.2 | 77.9              | 49.2 |
| greedy_dfs_sps              | 80.2       | 52.9 | 89.3      | 79.7 | 55.7    | 34.4 | 61.6              | 37.3 |

Table 4.4: Trade Pooling Defense: Percentage of honest nodes selected as destination nodes to provide transaction services at the end of the simulation.

| Malicious Rate ( $\alpha$ ) | Normal Run |      | Collusion |      | Slander |      | Slander-Collusion |      |
|-----------------------------|------------|------|-----------|------|---------|------|-------------------|------|
|                             | 0.20       | 0.51 | 0.20      | 0.51 | 0.20    | 0.51 | 0.20              | 0.51 |
| max_flow                    | 94.6       | 81.7 | 71.5      | 27.5 | 85.0    | 44.3 | 62.6              | 9.2  |
| ppr                         | 96.1       | 86.3 | 85.6      | 42.1 | 66.5    | 18.3 | 52.3              | 11.9 |
| ping_pong_dfs_ppr_stge      | 91.6       | 77.5 | 51.1      | 6.9  | 93.9    | 76.3 | 78.6              | 10.2 |
| ping_pong_dfs_ppr_socs      | 90.1       | 80.2 | 58.6      | 6.1  | 93.4    | 77.8 | 74.9              | 5.3  |
| shortest_path               | 92.7       | 79.6 | 66.2      | 29.2 | 86.2    | 57.7 | 55.5              | 10.7 |
| greedy_dfs                  | 91.6       | 76.7 | 83.8      | 45.2 | 81.2    | 47.5 | 75.9              | 35.3 |
| befs_nonrs                  | 92.4       | 80.3 | 80.5      | 37.8 | 84.6    | 43.6 | 73.8              | 36.0 |
| befs                        | 91.2       | 76.5 | 79.9      | 40.3 | 84.3    | 46.3 | 75.9              | 37.6 |
| ping_pong_dfs_maxflow_stge  | 78.5       | 55.8 | 51.9      | 17.4 | 95.3    | 77.4 | 92.7              | 22.9 |
| ping_pong_dfs_maxflow_socs  | 80.0       | 60.4 | 55.9      | 15.8 | 96.7    | 81.7 | 91.1              | 17.8 |
| greedy_dfs_sps              | 40.7       | 19.3 | 43.3      | 27.7 | 64.7    | 50.4 | 67.5              | 65.9 |

Table 4.5: Trade Pooling Defense: Percentage of honest nodes cheated during transactions at the end of the simulation.

| Malicious Rate ( $\alpha$ ) | Normal Run |      | Collusion |      | Slander |      | Slander-Collusion |      |
|-----------------------------|------------|------|-----------|------|---------|------|-------------------|------|
|                             | 0.20       | 0.51 | 0.20      | 0.51 | 0.20    | 0.51 | 0.20              | 0.51 |
| max_flow                    | 4.0        | 6.8  | 18.8      | 26.4 | 6.1     | 15.0 | 21.1              | 28.7 |
| ppr                         | 3.1        | 5.2  | 8.3       | 18.2 | 15.5    | 23.4 | 22.6              | 26.5 |
| ping_pong_dfs_ppr_stge      | 5.7        | 7.4  | 24.5      | 29.1 | 3.2     | 4.3  | 4.9               | 27.6 |
| ping_pong_dfs_ppr_socs      | 6.6        | 8.0  | 21.8      | 29.7 | 2.4     | 3.2  | 5.3               | 26.2 |
| shortest_path               | 5.0        | 7.5  | 19.8      | 24.5 | 6.7     | 8.2  | 20.1              | 27.4 |
| greedy_dfs                  | 5.7        | 8.7  | 5.7       | 9.8  | 6.1     | 8.3  | 6.1               | 10.6 |
| befs_nonrs                  | 5.1        | 7.7  | 6.4       | 10.9 | 6.1     | 9.4  | 6.5               | 10.6 |
| befs                        | 5.9        | 8.8  | 6.4       | 10.0 | 6.2     | 8.4  | 6.5               | 10.4 |
| ping_pong_dfs_maxflow_stge  | 13.0       | 16.1 | 27.6      | 24.2 | 2.0     | 3.9  | 3.6               | 21.2 |
| ping_pong_dfs_maxflow_socs  | 12.1       | 15.1 | 27.9      | 23.6 | 1.8     | 3.7  | 4.3               | 23.0 |
| greedy_dfs_sps              | 36.4       | 32.7 | 42.0      | 48.8 | 22.4    | 19.8 | 24.3              | 22.1 |

Table 4.6: Trade Pooling Defense: Percentage of malicious nodes cheated during transactions at the end of the simulation.

| Malicious Rate ( $\alpha$ ) | Normal Run |      | Collusion |      | Slander |      | Slander-Collusion |      |
|-----------------------------|------------|------|-----------|------|---------|------|-------------------|------|
|                             | 0.20       | 0.51 | 0.20      | 0.51 | 0.20    | 0.51 | 0.20              | 0.51 |
| max_flow                    | 1.3        | 7.7  | 0.3       | 0.3  | 23.2    | 44.9 | 14.7              | 1.8  |
| ppr                         | 1.1        | 5.8  | 0.3       | 0.3  | 22.0    | 41.3 | 6.2               | 1.2  |
| ping_pong_dfs_ppr_stge      | 1.8        | 10.0 | 0.1       | 0.0  | 19.3    | 43.8 | 6.6               | 1.6  |
| ping_pong_dfs_ppr_socs      | 2.1        | 7.7  | 0.2       | 0.0  | 18.9    | 45.6 | 6.6               | 1.0  |
| shortest_path               | 1.7        | 8.5  | 0.3       | 0.3  | 23.4    | 46.2 | 6.2               | 1.2  |
| greedy_dfs                  | 1.7        | 9.4  | 0.2       | 0.1  | 22.3    | 41.2 | 6.4               | 2.1  |
| befs_nonrs                  | 1.7        | 8.0  | 0.2       | 0.1  | 15.6    | 39.2 | 6.1               | 2.2  |
| befs                        | 1.9        | 9.7  | 0.2       | 0.1  | 17.5    | 41.3 | 6.5               | 2.3  |
| ping_pong_dfs_maxflow_stge  | 3.9        | 17.4 | 0.2       | 0.0  | 23.3    | 45.4 | 20.6              | 4.9  |
| ping_pong_dfs_maxflow_socs  | 3.6        | 15.3 | 0.3       | 0.1  | 21.4    | 46.8 | 21.0              | 3.5  |
| greedy_dfs_sps              | 8.1        | 27.6 | 0.1       | 0.2  | 42.7    | 59.6 | 37.9              | 58.0 |

**GREEDY\_DFS** emerge as the best algorithms, showing the highest rates of honest destination selection among all embedding-based metrics. The most interesting result occurs at a malicious rate of  $\alpha = 0.51$ , where these algorithms even outperform global algorithms such as **MAXFLOW** and **SHORTEST\_PATH**.

Furthermore, examining the cheated transaction table for honest nodes (Table 4.5) shows that these same three algorithms are the only ones, including global metrics, to stay below an 11% threshold for cheated honest nodes, with **GREEDY\_DFS** achieving the best result at 9.8%. In contrast, all remaining algorithms exceed 23%, reaching up to 30%. However, observing the source node metrics confirms that initiating nodes are never excluded from the system, proving that even under attack, honest nodes are cheated only roughly one out of three times, meaning the defense still yields notable benefits in worst case scenarios. Nevertheless, for algorithms above the 23% threshold, this defense introduces severe honest destination starvation. Specifically, **PINGPONG\_DFS\_PPR\_STGE** and **PINGPONG\_DFS\_PPR\_SOCS** exhibit honest destination selection values of only 6.9% and 6.1%, respectively. Consequently, these findings confirm that only **GREEDY\_DFS**, **BEFS\_NONRS**, and **BEFS** provide satisfactory protection, whereas alternative metrics entail critical security risks.

Regarding the slander attack (Section 4.3.5.2), these end-of-simulation metrics provide a much clearer picture compared to our previous temporal analyses. Specifically, the results show that the attack fails to isolate honest nodes from participating in the network.

As shown in the source node table (Table 4.3), even at the highest malicious rate ( $\alpha = 0.51$ ), honest source participation remains close to the normal run scenario, confirming that honest nodes are not excluded. Furthermore, honest destination selection (Table 4.4) yields even higher values than under collusion, demonstrating that candidate pooling is particularly effective against slander.

The cumulative honest transaction plots examined earlier were partially distorted by false ratings submitted by malicious nodes. While Table 4.6 suggests that malicious nodes are heavily affected, these figures are skewed by false reports and could lead to an incorrect global interpretation. Instead, examining the cheated honest node metrics (Table 4.5) shows that slander has minimal real impact, with results very close to the baseline run.

In this scenario, PingPong algorithms perform best. For cheated honest transactions, **PINGPONG\_DFS\_PPR\_SOCS** achieves the best result at 3.2%. For honest destination selection, PingPong variants range from a minimum of 76.3% (**PINGPONG\_DFS\_PPR\_STGE**) to a maximum of 81.7% (**PINGPONG\_DFS\_MAXFLOW\_SOCS**). Trailing further behind are the usual **BEFS\_NONRS**, **BEFS**, and **GREEDY\_DFS**, ranging between 43.6% and 47.5%, while **GREEDY\_DFS\_SPS** once again shows the worst performance. Consequently, for this attack vector, the defense mechanism fully achieves its objective by effectively reducing cheated honest transactions while preventing potential starvation issues, in contrast to the collusion attack. This holds true across all analyzed embedding metrics, with the logical exception

of `GREEDY_DFS_SPS`.

To conclude this first evaluation, we analyze the combined slander-collusion scenario (Section 4.3.5.4). As noted previously, candidate pooling performs well at low malicious rates, but significant issues emerge when the malicious rate rises to  $\alpha = 0.51$ .

Indeed, examining honest destination selection (Table 4.4) shows that most algorithms drop sharply. The main exceptions are `GREEDY_DFS`, `BEFS_NONRS`, and `BEFS`, which maintain an average above 35%. Most remaining embedding algorithms reach a maximum of only 10%, with `PINGPONG_DFS_MAXFLOW_STGE` and `PINGPONG_DFS_MAXFLOW_SOCS` as the only exceptions reaching 22.9% and 17.8%, respectively. This confirms our earlier finding that candidate pooling struggles under combined attacks, leading to transaction starvation for honest destination nodes.

On a positive note, honest source participation (Table 4.3) remains stable, ensuring that honest nodes are not excluded from initiating trades. Furthermore, the cheated honest node metrics (Table 4.5) show that despite destination starvation, the rate of cheated honest trades stays around 10% for `GREEDY_DFS`, `BEFS_NONRS`, and `BEFS`, whereas all other embedding algorithms reach at least 21%, remaining once again below the thresholds of global metrics. Nevertheless, this scenario mirrors the pattern previously described under collusion attacks. While these alternative embedding metrics could technically be considered viable, destination starvation introduces a critical security risk. Consequently, once again, only `GREEDY_DFS`, `BEFS_NONRS`, and `BEFS` achieve satisfactory performance.

Drawing a final conclusion from these evaluations, all analyzed metrics perform reasonably well under baseline conditions. However, when considering all three attack scenarios, `BEFS_NONRS`, `BEFS`, and `GREEDY_DFS` emerge as the most effective algorithms. Across every threat vector, they maintain consistent performance by keeping honest cheated transactions low, even at high malicious rates. Furthermore, they preserve steady destination selection for honest nodes, holding around 35%, which partially mitigates honest destination starvation. In contrast, PingPong algorithms exhibit volatile behavior, ranging from optimal results in specific scenarios to severe degradation under collusion or combined attacks, while `GREEDY_DFS_SPS` is likewise discarded due to its consistently poor efficiency across all tests. For these reasons, neither PingPong variants nor `GREEDY_DFS_SPS` can be considered viable choices for robust deployment. In a dynamic threat landscape, maintaining reliable baseline protection across all potential attack vectors is far more critical than excelling in isolated scenarios while collapsing under others. Consequently, for the top performing algorithms, candidate pooling proves to be a noninvasive defense mechanism that provides tangible benefits to overall metric performance. Most importantly, under collusion and slander collusion scenarios, these algorithms achieve significantly superior overall results compared to global metrics such as `SHORTEST_PATH` and `MAX_FLOW`, further elevating their potential for secure deployment.

#### 4.3.9.2 Single Punishment Defense

Finally, we evaluate the Single Punishment defense mechanism. Table 4.7 reports the percentage of honest nodes initiating transactions, while Table 4.8 presents the percentage of honest nodes selected as destination targets to provide the service. Additionally, Table 4.9 shows the percentage of honest nodes that get cheated, and Table 4.10 reports the percentage of cheated malicious nodes.

The configuration used for this scenario is (`--punishment single`) and (`--poolTrade 5`).

Table 4.7: Single Punishment Defense: Percentage of honest nodes initiating transactions as source nodes at the end of the simulation.

| Malicious Rate ( $\alpha$ ) | Normal Run |      | Collusion |      | Slander |      | Slander-Collusion |      |
|-----------------------------|------------|------|-----------|------|---------|------|-------------------|------|
|                             | 0.20       | 0.51 | 0.20      | 0.51 | 0.20    | 0.51 | 0.20              | 0.51 |
| max_flow                    | 74.5       | 47.3 | 75.4      | 45.8 | 83.5    | 69.4 | 76.3              | 44.9 |
| ppr                         | 74.5       | 45.7 | 75.4      | 45.8 | 74.5    | 45.7 | 75.4              | 45.8 |
| ping_pong_dfs_ppr_stge      | 93.4       | 84.1 | 82.0      | 51.2 | 93.2    | 58.4 | 69.1              | 37.3 |
| ping_pong_dfs_ppr_soc       | 76.8       | 50.1 | 76.7      | 49.8 | 93.9    | 80.2 | 77.0              | 45.1 |
| shortest_path               | 74.5       | 45.7 | 75.4      | 45.8 | 90.5    | 79.4 | 78.5              | 47.3 |
| greedy_dfs                  | 94.8       | 89.0 | 77.2      | 46.8 | 91.1    | 58.9 | 66.4              | 25.3 |
| befs_nonrs                  | 75.6       | 46.1 | 75.6      | 46.5 | 95.5    | 83.2 | 79.4              | 47.6 |
| befs                        | 94.2       | 88.9 | 76.9      | 49.6 | 92.0    | 55.8 | 63.7              | 29.6 |
| ping_pong_dfs_maxflow_stge  | 95.0       | 85.4 | 84.3      | 59.4 | 84.0    | 47.9 | 73.1              | 41.9 |
| ping_pong_dfs_maxflow_soc   | 77.9       | 50.8 | 76.3      | 49.3 | 84.9    | 71.9 | 74.5              | 46.0 |
| greedy_dfs_sps              | 98.6       | 90.4 | 97.1      | 79.3 | 95.4    | 65.9 | 34.7              | 3.1  |

For the normal run analyzed in Section 4.3.6, the primary drawback across almost all metrics, including BEFS and GREEDY\_DFS, is a sharp drop in Pathfinding Success Rates. The only algorithms capable of resisting this decline and maintaining high pathfinding performance are BEFS\_NONRS, PINGPONG\_DFS\_MAXFLOW\_SPCS, and PINGPONG\_DFS\_PPR\_SPCS. This pathfinding degradation stems from the strict nature of the single punishment defense, which tends to fragment the network topology. Consequently, the overall evaluation of these metrics is heavily dictated by this factor.

Interestingly, regarding honest source participation (Table 4.7), algorithms such as GREEDY\_DFS, BEFS, PINGPONG\_DFS\_PPR\_STGE, and PINGPONG\_DFS\_MAXFLOW\_STGE achieve over 93% honest source representation. This yields roughly a 20% increase at a malicious rate of  $\alpha = 0.20$  and a 40% increase at  $\alpha = 0.51$ , demonstrating that the defense functions effectively for them. However, as previously noted, this high filtering efficiency comes at the cost of graph fragmentation given the small network size. Conversely, observing the

Table 4.8: Single Punishment Defense: Percentage of honest nodes selected as destination nodes to provide transaction services at the end of the simulation.

| Malicious Rate ( $\alpha$ ) | Normal Run |      | Collusion |      | Slander |      | Slander-Collusion |      |
|-----------------------------|------------|------|-----------|------|---------|------|-------------------|------|
|                             | 0.20       | 0.51 | 0.20      | 0.51 | 0.20    | 0.51 | 0.20              | 0.51 |
| max_flow                    | 99.2       | 96.0 | 97.0      | 86.6 | 95.5    | 79.2 | 76.7              | 23.9 |
| ppr                         | 99.1       | 94.1 | 98.1      | 92.0 | 96.4    | 79.0 | 77.8              | 32.3 |
| ping_pong_dfs_ppr_stge      | 98.8       | 95.0 | 94.3      | 73.2 | 96.1    | 64.0 | 70.4              | 32.3 |
| ping_pong_dfs_ppr_socs      | 99.1       | 96.9 | 90.9      | 76.8 | 97.4    | 83.2 | 76.8              | 36.4 |
| shortest_path               | 99.0       | 95.1 | 94.9      | 77.7 | 97.9    | 86.4 | 77.2              | 31.2 |
| greedy_dfs                  | 99.0       | 95.9 | 91.1      | 72.9 | 94.2    | 58.9 | 64.8              | 15.1 |
| befs_nonrs                  | 99.2       | 97.0 | 90.5      | 74.1 | 97.7    | 77.7 | 77.5              | 35.1 |
| befs                        | 98.6       | 95.9 | 92.3      | 76.4 | 95.5    | 58.9 | 61.9              | 19.6 |
| ping_pong_dfs_maxflow_stge  | 98.9       | 94.6 | 94.6      | 73.5 | 95.6    | 68.3 | 76.7              | 31.6 |
| ping_pong_dfs_maxflow_socs  | 98.8       | 96.6 | 93.3      | 75.6 | 96.8    | 79.4 | 78.5              | 36.9 |
| greedy_dfs_sps              | 97.2       | 81.8 | 92.8      | 62.3 | 96.1    | 56.4 | 45.5              | 5.3  |

Table 4.9: Single Punishment Defense: Percentage of honest nodes cheated during transactions at the end of the simulation.

| Malicious Rate ( $\alpha$ ) | Normal Run |      | Collusion |      | Slander |      | Slander-Collusion |      |
|-----------------------------|------------|------|-----------|------|---------|------|-------------------|------|
|                             | 0.20       | 0.51 | 0.20      | 0.51 | 0.20    | 0.51 | 0.20              | 0.51 |
| max_flow                    | 1.4        | 1.8  | 2.6       | 5.1  | 2.0     | 6.5  | 6.8               | 19.6 |
| ppr                         | 1.4        | 2.6  | 1.5       | 2.8  | 2.4     | 5.6  | 3.7               | 11.3 |
| ping_pong_dfs_ppr_stge      | 1.9        | 3.1  | 3.7       | 7.1  | 3.3     | 9.1  | 2.9               | 5.2  |
| ping_pong_dfs_ppr_socs      | 1.6        | 1.5  | 2.7       | 5.9  | 2.4     | 6.5  | 3.4               | 7.4  |
| shortest_path               | 1.5        | 2.1  | 2.0       | 4.6  | 2.2     | 5.2  | 4.8               | 13.4 |
| greedy_dfs                  | 1.8        | 2.7  | 1.8       | 3.3  | 3.7     | 12.0 | 4.1               | 8.3  |
| befs_nonrs                  | 1.6        | 1.4  | 1.9       | 3.4  | 2.3     | 9.8  | 3.8               | 10.0 |
| befs                        | 2.0        | 2.7  | 1.9       | 3.4  | 3.3     | 10.9 | 4.0               | 8.4  |
| ping_pong_dfs_maxflow_stge  | 1.9        | 3.1  | 3.5       | 8.2  | 3.0     | 7.6  | 2.7               | 9.1  |
| ping_pong_dfs_maxflow_socs  | 1.6        | 1.5  | 3.9       | 7.0  | 2.2     | 7.3  | 2.7               | 8.2  |
| greedy_dfs_sps              | 2.8        | 8.1  | 5.8       | 15.4 | 3.4     | 17.1 | 3.8               | 0.8  |

Table 4.10: Single Punishment Defense: Percentage of malicious nodes cheated during transactions at the end of the simulation.

| Malicious Rate ( $\alpha$ ) | Normal Run |      | Collusion |      | Slander |      | Slander-Collusion |      |
|-----------------------------|------------|------|-----------|------|---------|------|-------------------|------|
|                             | 0.20       | 0.51 | 0.20      | 0.51 | 0.20    | 0.51 | 0.20              | 0.51 |
| max_flow                    | 0.5        | 2.0  | 0.4       | 0.7  | 15.6    | 27.1 | 8.5               | 5.1  |
| ppr                         | 0.5        | 2.7  | 0.4       | 0.7  | 25.0    | 50.3 | 6.2               | 1.2  |
| ping_pong_dfs_ppr_stge      | 0.2        | 1.0  | 0.2       | 0.4  | 6.5     | 33.8 | 4.1               | 1.5  |
| ping_pong_dfs_ppr_socs      | 0.4        | 1.7  | 0.3       | 0.5  | 5.8     | 17.0 | 3.0               | 0.8  |
| shortest_path               | 0.6        | 2.4  | 0.3       | 0.6  | 9.2     | 18.4 | 3.9               | 1.3  |
| greedy_dfs                  | 0.2        | 0.8  | 0.2       | 0.4  | 8.2     | 33.6 | 2.9               | 1.0  |
| befs_nonrs                  | 0.5        | 1.7  | 0.2       | 0.5  | 4.2     | 13.7 | 1.8               | 0.7  |
| befs                        | 0.2        | 0.9  | 0.3       | 0.5  | 7.4     | 35.9 | 2.6               | 1.2  |
| ping_pong_dfs_maxflow_stge  | 0.2        | 1.3  | 0.2       | 0.4  | 15.4    | 45.9 | 5.9               | 1.4  |
| ping_pong_dfs_maxflow_socs  | 0.6        | 2.0  | 0.3       | 0.6  | 14.6    | 24.7 | 6.2               | 1.4  |
| greedy_dfs_sps              | 0.2        | 1.8  | 0.0       | 0.1  | 4.3     | 27.0 | 17.6              | 4.4  |

metrics with higher pathfinding rates reveals that their honest source participation remains essentially comparable to the values observed under candidate pooling across both low and high malicious rates.

Examining honest destination selection (Table 4.8) shows that at a malicious rate of  $\alpha = 0.20$ , nearly all algorithms achieve approximately 99% honest destination representation, while at  $\alpha = 0.51$ , the lowest value across all metrics remains above 94%. This represents a substantial overall improvement, particularly for algorithms like PINGPONG\_DFS\_MAXFLOW, which see at least a 40% increase in honest destinations. In particular, BEFS\_NONRS captures the highest percentage of honest destination nodes, closely followed by PINGPONG\_DFS\_MAXFLOW\_SOCS and PINGPONG\_DFS\_PPR\_SOCS.

Finally, analyzing cheated honest transactions (Table 4.9) in comparison with candidate pooling (Table 4.5) highlights substantial reductions. Although baseline values under pooling were already low, metrics maintaining higher pathfinding exhibit remarkable decreases. Specifically, at  $\alpha = 0.51$ , BEFS\_NONRS suffers only 1.4% cheated transactions (down 6.6% compared to pooling alone), PINGPONG\_DFS\_MAXFLOW\_SOCS drops to 1.5% (compared to 15.1%), and PINGPONG\_DFS\_PPR\_SOCS reaches 1.5% (compared to 8.0%). In particular, these algorithms once again outperform global metrics in mitigating dishonest transactions targeting honest nodes. Excluding the inherently flawed GREEDY\_DFS\_SPS, all remaining algorithms fall below 3.1% cheated honest transactions, which is a remarkably solid result given a context where malicious nodes constitute the majority.

A final comparison between cheated malicious transactions under pooling (Table 4.6) and

under single punishment (Table 4.10) demonstrates a significant drop in cheated trades for malicious nodes as well. Re-evaluating this alongside honest node metrics confirms that malicious nodes are rarely selected as top trade candidates due to the overall suppression of cheated transactions, thereby making the network fundamentally more secure and supporting the earlier observations from the baseline normal run.

Ultimately, deploying the remaining metrics requires deciding whether the risk of network fragmentation is acceptable. If so, `BEFS_NONRS`, `PINGPONG_DFS_MAXFLOW_SOCS`, and `PINGPONG_DFS_PPR_SOCS` represent optimal solutions, effectively addressing the threat without introducing retroactive side effects or performance tradeoffs. Consequently, these findings demonstrate that the defense mechanism yielded a substantial overall improvement in this context.

Moving to collusion attacks, pathfinding degradation was previously absent due to higher network connectivity resulting from reputation boosts. Analyzing honest source participation under single punishment (Table 4.7) reveals little difference compared to candidate pooling alone (Table 4.3), except for a slight increase in PingPong variants.

The most compelling evidence lies in honest destination selection (Table 4.8), where all algorithms achieve a minimum increase of 15% at a malicious rate of  $\alpha = 0.20$ . In particular, at  $\alpha = 0.51$ , every algorithm demonstrates an increase of approximately 30%. As discussed in Section 4.3.6, this confirms that the defense performs remarkably well across all metrics.

This efficacy is further corroborated by cheated honest transactions (Table 4.9). While candidate pooling alone yielded an optimal cheated transaction rate of approximately 10% with `BEFS`, single punishment reduces the maximum rate across all metrics to 8% excluding `GREEDY_DFS_SPS`, with the best result achieved by `GREEDY_DFS` at 3.3%, closely followed by `BEFS_NONRS` and `BEFS` at 3.4%, as these embedding algorithms once again surpass global metrics such as `MAX_FLOW` and `SHORTEST_PATH`.

Consequently, single punishment proves to be a highly effective defense against collusion attacks. It successfully resolves the vulnerabilities previously observed under candidate pooling alone, establishing itself as a reliable defense across all evaluated metrics.

Turning to slander attacks, as previously highlighted in Section 4.3.6, pathfinding success drops significantly under single punishment, a degradation that becomes particularly pronounced at a malicious rate of  $\alpha = 0.51$ . In this setting, only `BEFS_NONRS`, `PINGPONG_DFS_PPR_SOCS`, and `PINGPONG_DFS_MAXFLOW_SOCS` manage to sustain Pathfinding Success Rates slightly above 50%. Given this severe constraint, the defense appears fundamentally flawed for pure slander attacks on a broader scale, as it inflicts excessive damage on graph connectivity. Nevertheless, analyzing these three resilient algorithms provides essential insights into the underlying trade-offs.

Examining honest source participation (Table 4.7) in comparison to candidate pooling (Table 4.3) reveals notable gains at  $\alpha = 0.20$ , with PingPong variants increasing by 10% and BEFS\_NONRS by 13%. At  $\alpha = 0.51$ , these improvements become even more pronounced, reaching a 34% increase for BEFS\_NONRS, 40% for PINGPONG\_DFS\_PPR\_SOCS, and 26% for PINGPONG\_DFS\_MAXFLOW\_SOCS. As expected from the overall pathfinding decline, these metrics demonstrate that malicious nodes are effectively isolated, thereby ensuring that active path initiation remains largely driven by honest nodes.

A similar positive trend emerges in honest destination selection (Table 4.8) at  $\alpha = 0.51$ , where BEFS\_NONRS achieves a 44% increase relative to candidate pooling, thereby mitigating destination starvation, while PingPong SoCS variants preserve their previously high baseline performance. Regarding cheated honest transactions (Table 4.9), figures remain virtually identical to candidate pooling, showing minor variations of 2% to 3% and staying strictly below 10%. Conversely, cheated malicious transactions (Table 4.10) experience a substantial reduction, dropping by 26% for BEFS\_NONRS, 28% for PINGPONG\_DFS\_PPR\_SOCS, and 22% for PINGPONG\_DFS\_MAXFLOW\_SOCS. This sharp decrease proves that malicious nodes are rapidly severed from interaction paths once their dishonest feedback is detected.

Synthesizing these observations leads to a critical verdict. Single punishment offers the distinct advantage of isolating malicious actors by penalizing false testimonies, but it does so at the cost of severe network fragmentation. With only three algorithms maintaining even a 50% Pathfinding Success Rate, the defense proves overly punitive for pure slander scenarios across the majority of metrics. While BEFS\_NONRS exhibits substantial overall performance gains, PingPong SoCS variants remain essentially unchanged in performance. Consequently, aside from excluding malicious actors, deploying the defense under slander attacks provides minimal comparative difference for these two PingPong variants relative to candidate pooling alone. Nevertheless, taking the broader picture and prior evaluations into account, accepting network fragmentation under PingPong SoCS variants remains justified because single punishment resolves critical vulnerabilities present under candidate pooling for other threat vectors. Meanwhile, for BEFS\_NONRS, the defense introduces substantial overall improvements, establishing it as an exceptionally strong candidate for single punishment deployment across all threat scenarios.

In light of the preceding conclusions, the evaluation focuses exclusively on BEFS\_NONRS, PINGPONG\_DFS\_PPR\_SOCS, and PINGPONG\_DFS\_MAXFLOW\_SOCS to assess whether they maintain strong performance under slander-collusion attacks.

Examining honest source participation (Table 4.7) at a malicious rate of  $\alpha = 0.51$  reveals minimal shifts, with a slight decrease of 3% to 4% for PINGPONG\_DFS\_PPR\_SOCS and PINGPONG\_DFS\_MAXFLOW\_SOCS, while BEFS\_NONRS remains virtually unchanged.

As hypothesized in Section 4.3.6, destination starvation persists under this threat vector. However, notable gains emerge in honest destination selection: PINGPONG\_DFS\_PPR\_SOCS

increases from 5% under candidate pooling to 36% under single punishment, and `PINGPONG_DFS_MAXFLOW_SOCS` rises from 18% to 37%, whereas `BEFS_NONRS` holds essentially steady.

Furthermore, analyzing cheated honest transactions (Table 4.9) highlights marked improvements for the PingPong variants. Specifically, `PINGPONG_DFS_MAXFLOW_SOCS` drops from 23% to 8% and `PINGPONG_DFS_PPR_SOCS` falls from 26% to 7%, while `BEFS_NONRS` maintains its previous baseline. Crucially, all three analyzed embedding metrics once again achieve superior results compared to global metrics. Simultaneously, these findings confirm that the defense mechanism yields modest performance gains across the evaluated metrics, representing a partial success while ensuring no additional degradation compared to baseline candidate pooling.

Synthesizing these findings across all evaluated threat vectors, the overall analysis demonstrates that combining single punishment with candidate pooling yields performance gains during baseline normal runs and particularly against collusion attacks across all metrics. However, severe limitations arise under slander attacks, where a high malicious rate induces network fragmentation. The few exceptions are `BEFS_NONRS`, `PINGPONG_DFS_PPR_SOCS`, and `PINGPONG_DFS_MAXFLOW_SOCS`, which nevertheless sustain Pathfinding Success Rates only slightly above 50%. While these three algorithms deliver localized gains, these improvements are not substantial enough to justify the risk of network fragmentation. Similarly, under slander-collusion attacks, single punishment fails to resolve the issue of destination starvation.

Furthermore, while punishment aims to remove malicious actors from the system, it is worth questioning whether completely expelling or disconnecting them is necessary when candidate pooling alone already maintains a consistently low percentage of cheated honest nodes. From this, candidate pooling represents the best overall solution, whereas single punishment does not prove worthwhile due to the operational risks it introduces under specific threat vectors.

Ultimately, the three top-performing algorithms throughout this study are `BEFS_NONRS`, `BEFS`, and `GREEDY_DFS`, as they were the only metrics to maintain consistently above-average performance across all evaluated scenarios while preserving acceptable operational values. This makes them the sole metrics currently capable of managing diverse threat vectors under high malicious rates in a global context, with `BEFS_NONRS` earning top distinction as the sole algorithm among the primary trio capable of withstanding the stringent constraints of single punishment. Importantly, these same three algorithms frequently achieve superior performance compared to traditional global metrics such as `MAX_FLOW` and `SHORTEST_PATH`, positioning them as a promising foundation for future research into novel embedding techniques, defense mechanisms, and analytical frameworks for distributed systems.

# Chapter 5

## Conclusion

The goal of this thesis was to evaluate the security and resilience of the embedding metrics proposed by Romano. To achieve this, we built a discrete-event simulator incorporating various parameters, such as attack strategies, defense mechanisms, and varying malicious rates, which allowed us to analyze the metrics from multiple perspectives. Specifically, we used time-series plots to observe metric behavior over time, alongside end-of-test transaction data to better understand how attacks and defenses impacted each metric.

By analyzing all the gathered data, we reached the conclusion that candidate pooling represents the most effective defense mechanism evaluated in this study, as it maintains a non-invasive profile while providing robust resilience, unlike single punishment, which induces severe network fragmentation under specific threat vectors. Under candidate pooling, we successfully reduced the proportion of cheated honest nodes to very low levels while ensuring that selected destinations were predominantly honest rather than malicious. This significantly improved overall metric performance across nearly all evaluated embedding algorithms and effectively mitigated the aggregate damage caused by attacks. Among them, BeFS, Greedy DFS, and BeFS-NoNRS emerge as the top three performing algorithms, with BeFS-NoNRS achieving the highest overall distinction as the sole metric capable of preventing topological fragmentation even under single punishment. Notably, these localized embedding metrics frequently outperformed traditional global algorithms such as Shortest Path and Max Flow, highlighting their underlying characteristics as a promising foundation for future research into novel trust metrics and defense strategies. Consequently, the developed simulation environment successfully fulfilled the primary objective of this thesis, offering a comprehensive, multi-dimensional evaluation and delivering deep analytical insights into the behavior of decentralized reputation systems.

Nevertheless, this study presents a few limitations, primarily stemming from real-world execution time constraints. To keep runtimes manageable, experiments were conducted

on a small network of 100 nodes with a low join rate, as scaling the topology significantly increased computational overhead. These same temporal restrictions also constrained the scope of the evaluation, allowing for only a relatively limited selection of defense mechanisms and attack strategies to be implemented. Furthermore, calibrating simulation parameters, exploring late-stage data ideas, or resolving post-run errors consumed considerable time, leaving minimal margin for iterative adjustments. Crucially, testing within a reduced network topology inherently limits the generalizability of our findings. Because system dynamics in smaller settings do not always mirror large-scale environments, metrics that performed well in this setup might behave differently at scale, while those that struggled here could potentially achieve better results in larger networks.

These limitations and insights naturally point toward several promising directions for future research. Extending the evaluation to significantly larger networks and dynamic join rates would clarify how scaling and topological shifts affect metric behavior. Furthermore, future work could enrich the experimental framework by tuning the severity of defense mechanisms, introducing novel countermeasure paradigms, or designing new attack strategies specifically tailored to exploit candidate pooling. In parallel, incorporating more adaptive, intelligent behaviors for both honest and malicious nodes would provide a much closer approximation of real-world system dynamics. Ultimately, a dedicated investigation into BeFS, Greedy DFS, and BeFS-NoNRS would help isolate the precise structural properties enabling these localized metrics to consistently outperform global baselines such as Shortest Path and Max Flow, paving the way for a new generation of embedding techniques, resilient defenses, or specialized threat vectors.

# Bibliography

- [BP98] Sergey Brin and Lawrence Page. “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Computer Networks and ISDN Systems* 30.1–7 (1998), pp. 107–117.
- [CF05] Alice Cheng and Eric J. Friedman. “Sybilproof reputation mechanisms”. In: *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems*. ACM, 2005, pp. 128–132.
- [Cla+01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. “Freenet: A Distributed Anonymous Information Storage and Retrieval System”. In: *International Workshop on Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 46–66.
- [Cor+01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 2nd. Cambridge, MA, USA: MIT Press, 2001.
- [DM08] George Danezis and Prateek Mittal. “SybilInfer: Detecting Sybil Nodes Using Social Networks”. In: *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*. ACM, 2008, pp. 1–11.
- [Del20] Matteo Dell’Amico. “Webs of Trust: Choosing Who to Trust on the Internet”. In: *Annual Privacy Forum*. Springer, 2020, pp. 113–126.
- [DC10] Matteo Dell’Amico and Licia Capra. “SoK: Sybil Defense in Wireless Networks and Social Systems”. In: *IEEE Transactions on Dependable and Secure Computing*. 2010.
- [Dij59] Edsger W. Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.
- [Dou02] John R. Douceur. “The Sybil Attack”. In: *International Workshop on Peer-to-Peer Systems (IPTPS)*. Springer, 2002, pp. 251–260.
- [EK72] Jack Edmonds and Richard M. Karp. “Theoretical improvements in algorithmic efficiency for network flow problems”. In: *Journal of the ACM* 19.2 (1972), pp. 248–264.

- [Fel+04] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. “Robust Incentive Techniques for Peer-to-Peer Networks”. In: *Proceedings of the 5th ACM Conference on Electronic Commerce (EC '04)*. ACM, 2004, pp. 102–111.
- [Fel+06] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. “Free-Riding and Whitewashing in Peer-to-Peer Systems”. In: *IEEE Journal on Selected Areas in Communications* 24.5 (2006), pp. 1010–1019.
- [FF56] Lester R. Ford and Delbert R. Fulkerson. “Maximal flow through a network”. In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404.
- [Fun11] Carol Fung. “Collaborative intrusion detection networks and insider attacks”. In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)* 2.1 (2011), pp. 63–74.
- [Gam90] Diego Gambetta. “Can We Trust Trust?”. In: *Trust: Making and Breaking Cooperative Relations*. Ed. by Diego Gambetta. Oxford: Basil Blackwell, 1990, pp. 213–238.
- [Hav02] Taher H. Haveliwala. “Topic-sensitive PageRank”. In: *Proceedings of the 11th International Conference on World Wide Web (WWW '02)*. ACM, 2002, pp. 517–528.
- [HZN09] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. “A Survey of Attack and Defense Techniques for Reputation Systems”. In: *ACM Computing Surveys* 42.1 (2009), 1:1–1:31.
- [JR22] Gordhan Jethava and Udai Pratap Rao. “User behavior-based and graph-based hybrid approach for detection of Sybil Attack in online social networks”. In: *Computer Communications* 186 (2022), pp. 24–37.
- [JIB07] Audun Jøsang, Roslan Ismail, and Colin Boyd. “A survey of trust and reputation systems for online service provision”. In: *Decision Support Systems* 43.2 (2007), pp. 618–644.
- [KLT11] Anne-Marie Kermarrec, Vincent Leroy, and Gilles Trédan. “Decentralized graph embedding for efficient social distance computation”. In: *2011 31st IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2011, pp. 423–432.
- [MC96] D. Harrison McKnight and Norman L. Chervany. *The Meanings of Trust*. MISRC Working Paper Series 96-04. University of Minnesota, Management Information Systems Research Center, 1996.
- [Rom25] Samuele Romano. “Building the Foundations for a Decentralized Reputation System”. MA thesis. Università degli Studi di Genova, Oct. 2025.

- [RBS16a] Stefanie Roos, Martin Beck, and Thorsten Strufe. “Anonymous addresses for efficient and resilient routing in f2f overlays”. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.
- [RBS16b] Stefanie Roos, Martin Beck, and Thorsten Strufe. “Voute-virtual overlays using tree embeddings”. In: *arXiv preprint arXiv:1601.06119* (2016).
- [Row+07] Hosam Rowaihy, William Enck, Patrick McDaniel, and Thomas La Porta. “An Admission Control Framework for Securing Peer-to-Peer Networks Against Sybil Attacks”. In: *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 2521–2525.
- [Sch01] Ruediger Schollmeier. “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications”. In: *Proceedings First International Conference on Peer-to-Peer Computing*. IEEE, 2001, pp. 101–102.
- [UL17] Oliver Ullrich and Daniel Lückcrath. “An Introduction to Discrete-Event Modeling and Simulation”. In: *Simulation Notes Europe 27.1* (2017), pp. 1–8.
- [Vel+08] Pedro B. Velloso, Rafael P. Laufer, Otto Carlos M. B. Duarte, and Guy Pujolle. “A Trust Model Robust to Slander Attacks in Ad Hoc Networks”. In: *Proceedings of the 17th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2008, pp. 1–6.
- [Yu+08] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. “Sybil-Limit: A Near-Optimal Social Network Defense Against Sybil Attacks”. In: *2008 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2008, pp. 3–17.
- [Yu+06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. “SybilGuard: Defending Against Sybil Attacks via Social Networks”. In: *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. ACM, 2006, pp. 267–278.
- [Zim95] Philip R. Zimmermann. *The Official PGP User’s Guide*. MIT Press, 1995.