UNIVERSITY OF ARKANSAS

# Validating Damage Assessment: A Simulation-Based Analysis of Blind Write Lineage in Fog Computing

Mariha Siddika Ahmad and Brajendra Panda

Presented by

Mariha Siddika Ahmad

Electrical Engineering and Computer Science Department

University of Arkansas

Fayetteville, AR 72701 USA

Email: ma135@uark.edu

IARIA

Mariha Siddika Ahmad

*Ph.D. Candidate in Computer Science, University of Arkansas*

•**Focus**:

Database security with expertise in damage assessment, recovery algorithms

•**Experience**:

Research Assistant, Database Security, University of Arkansas

Grading Assistant, AI, University of Arkansas

•**Publications**:

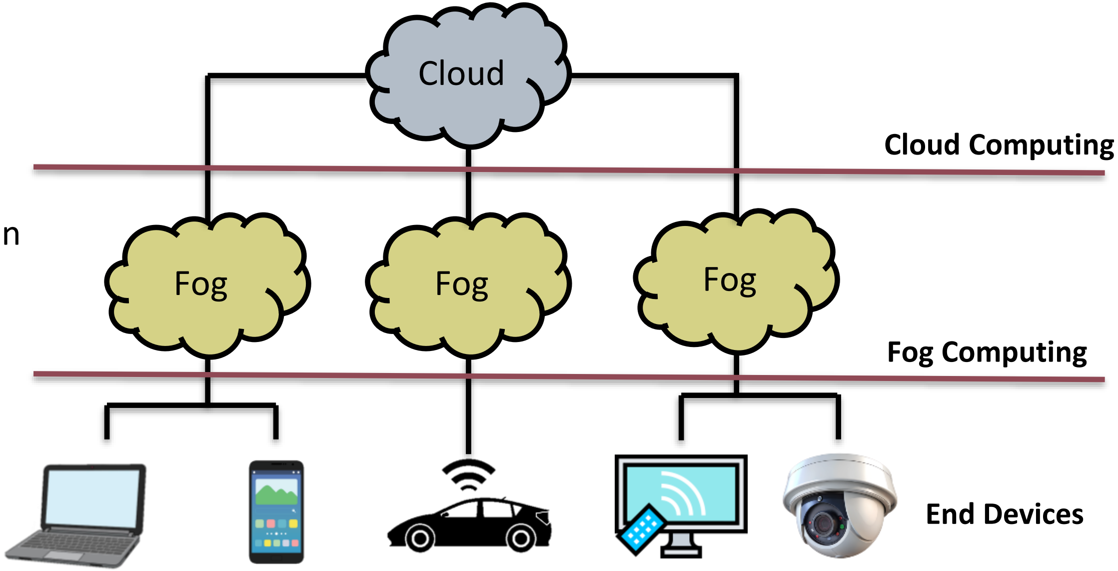
Damage Assessment in Fog Computing Systems: Developed a novel blind write lineage approach for secure IoT, presented at the SIoTEC Workshop 2024.

•**Projects**: Developed a malware classification tool using Pyramid Vision Transformer achieving 94.8% accuracy.

# Contents

# Fog Computing Overview

- Extension of cloud computing
- Brings computing resources closer to end users
- Key characteristics:
  - Low latency
  - Geographic distribution
  - Real-time interaction
  - Heterogeneity
- Benefits:
  - Reduced network congestion
  - Improved response times
  - Enhanced data security and privacy



# Problem Statement

**Vulnerability of Fog Systems to Cyberattacks**

- Fog computing extends cloud resources closer to users
- Inherits security risks from traditional cloud systems
- Interconnected nature creates larger attack surface
- Damage from attacks can propagate swiftly through the system

**Key Challenges:**

- Rapid damage assessment crucial for real-time operations
- Traditional recovery methods inadequate for fog environments
- Need for efficient damage containment and system restoration

**Impact:**

- Compromised data integrity across interconnected nodes
- Potential disruption of critical services (e.g., healthcare, emergency response)
- Financial and reputational risks for organizations

# Motivation

## Why Fog Systems Need Faster Recovery?

- **Critical Infrastructure Demands**
  - Emergency services, healthcare, and smart city systems require real-time functionality
  - Downtime can have severe consequences (e.g., delayed emergency response, compromised patient care)
- **Traditional Recovery Methods Inadequate**
  - System-wide shutdowns are too disruptive
  - Manual data checks are time-consuming and error-prone
- **Unique Challenges of Fog Computing**
  - Interconnected nodes allow rapid spread of compromised data
  - Heterogeneous systems increase complexity of damage assessment
  - Massive data volumes make traditional recovery methods impractical
- **Time is of the Essence**
  - Swift recovery prevents cascading failures across interconnected systems
  - Minimizes service disruptions and potential financial/reputational damage
  - Crucial for maintaining trust in critical infrastructure relying on fog computing

# Key Concepts

### Blind Writes

- Write operations that update data without reading existing values
- Characteristics:
  - ➢ No prior read request
  - ➢ Modification occurs regardless of original value
  - ➢ Absence of pre-write read operation

### Benefits

- Minimizes damage assessment time
- Accelerates damage recovery process
- Reduces system downtime during attacks

### Data Dependencies

- Relationships between data items tracked for damage assessment
- Types:
  - Direct dependencies (parent-child)
  - Indirect dependencies (ancestor-descendant)

### Importance

- Enables tracing of damage propagation
- Facilitates efficient isolation of compromised data
- Crucial for targeted recovery efforts

# Blind Write Lineage Model

**Blind Write Lineage:**

| The lineage starts with a blindly written data item (one that was updated without reading its previous value). | Subsequent data items in the lineage depend exclusively on either the initial blindly written item or its descendants. | This dependency can be direct or transitive (through other items in the chain). | While some items may use multiple data items for their update, at least one of those must be from the blind write lineage. |
|---|---|---|---|

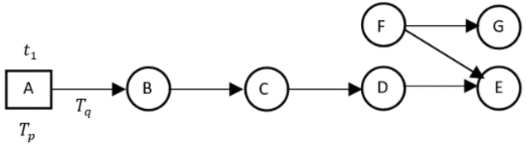



Figure 1: Blind Write Lineage

# Blind Write Lineage Model

The importance of this concept:

| | |
|---|---|
| Tracing damage | • It allows efficient tracing of potential damage propagation from a compromised blindly written item. |
| Focused assessment | • By identifying these lineages, the system can prioritize which data items need closer examination during damage assessment. |
| Efficiency | • It potentially reduces the scope of items that need to be checked, compared to examining all data dependencies. |

# Key components of the model

**Blind Write (BW) list**

- Records all blindly written data items
- Format: {[x, $t_x$, $T_x$] | x is blindly written, $t_x$ is update time and $T_x$ is the transaction }
- Purpose: Identify potential damage sources within a subgraph

**Blind Write Lineage (BW Lineage) list**

- Tracks dependencies of data items solely dependent on blind writes
- Format: [Parent_node → Child_node]
- Purpose: Trace potential damage propagation

**Subgraphs($G_i$)**

- Represent distinct clusters of related data items
- Each subgraph ($G_i$) contains its own:
  - Blind Write Set ($BWS_i$): Blindly written items in the subgraph
  - Children Data Set ($CDS_i$): Items dependent on $BWS_i$ elements

# Key components of the model

**Blind-Write Set($BWS_i$)**

- Contains blindly written items for a specific subgraph
- Format: {($x$, $t_x$) | *x is a parent node in* $G_i$ *where x is blindly written by a transaction and* $t_x$ *is the time x is updated*}
- Purpose: Identify potential damage sources within a subgraph

**Children Data Set($CDS_i$)**

- Contains data items dependent on $BWS_i$ items
- Format: {($y$, $t_y$)| *y is not blindly written, there exists at least one x in* $BWS_i$ *such that y is dependent on x* (*directly or transitively*) *and* $t_y$ *is the time when y is updated*}.
- Purpose: Track potential damage propagation within a subgraph

**Damaged set**

- Initially damaged data items by attacker
- Format: {($d_i$, $t_d$ )| *data item* $d_i$ *is written by an attacking transaction and therefore considered initially damaged,* $t_d$ *is the attack time*}
- Purpose: Starting point for damage assessment

# Key components of the model(Example)

| $G_i$ | $G_1$ | $G_2$ | $G_3$ |
|---|---|---|---|
| **$BWS_i$** | {(A,$t_1$), (X,$t_3$), (Y,$t_5$)} | {(P,$t_{10}$), (S,$t_{12}$)} | {(J,$t_{15}$)} |
| **$CDS_i$** | {(B,$t_2$), (C,$t_4$), (D,$t_6$), (E,$t_8$), (F,$t_7$), (G,$t_9$)} | {(Q,$t_{11}$),(C,$t_{14}$), (T,$t_{15}$)}, (R,$t_{18}$)} | {(C,$t_{16}$), (K,$t_{17}$)} |
| **D** | {(S,$t_{12}$)} | | |

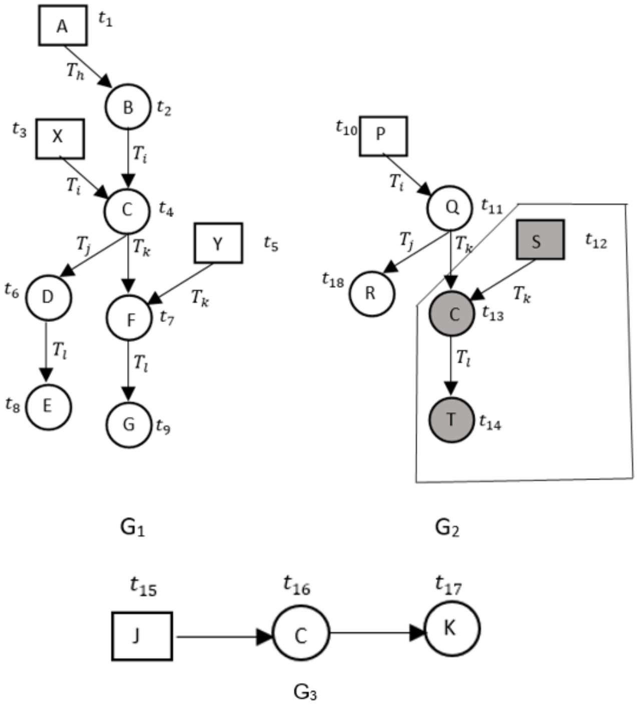



Figure 2: Multiple subgraphs in the data dependency (G)

# Cases of Blind Write Lineage Model

## Case 1: Single-Parent/Single-Child Lineage

- Data items are updated sequentially, each relying on a single predecessor.
- The lineage traces back to the original blindly written item.

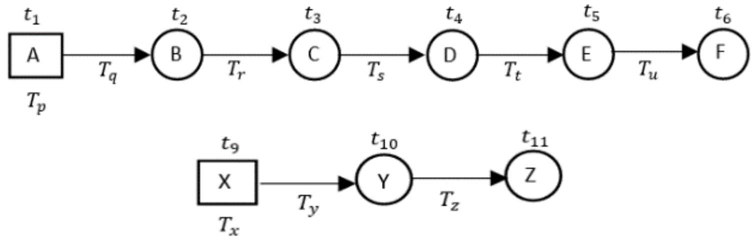



Figure 3: Single-Parent/Single-Child Lineage

| | |
|---|---|
| **BW list** | $[(A, t_1, T_p), (X, t_9, T_x)]$ |
| **BW lineage list** | [(A→B, B→C, C→D, D→E, E→F), (X→Y, Y→Z)] |

| **Key Points** |
|---|
| Simple and direct lineage. |
| Easy to trace damage propagation. |

# Cases of Blind Write Lineage Model

## Case 2: Multipath Lineage

- More complex scenario where a child node might have multiple parent nodes.
- Data items may be updated using multiple arguments.

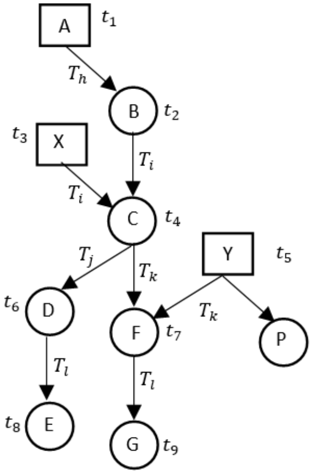



Figure 4: Complex Blind write Lineage.

| Key Points |
|---|
| Requires more refined damage assessment. |
| Complex dependencies necessitate careful tracing. |

| | |
|---|---|
| **BW list** | $[(A, t_1, T_a), (X, t_3, T_x), (Y, t_5, T_y)]]$ |
| **BW lineage list** | [(A→B, (B,X)→C, C→D, (C,Y)→F, D→E, F→G, Y→P)] |

# Damage Assessment

**Objective:**

- Quickly identify and isolate compromised data in fog computing systems

**Challenges:**

- Rapid propagation of damage
- Complex data dependencies
- Need for real-time recovery

**Key Concepts in Damage Assessment:**

- **Attack Time** ($t_a$):
  - Time when the malicious transaction occurred
  - Crucial for determining the timeline of damage.
- **Last Updated Time** ($t_{last\ updated\ time}$):
  - Last time each data item was updated
  - Helps in assessing whether an item was affected post-attack.
- **Affected Time** ($t_{aff}$):
  - Time when a data item was affected by the attack
  - Helps in assessing whether an item was affected post-attack.

# Damage Assessment

**Initialize Data Structures**

- Create Blind Write Set (**$BWS_i$**) and Children Data Set (**$CDS_i$**) for each subgraph.
- Maintain a Final Updated Time table for all data items.

**Identify Damaged Subgraphs**

- Use the initial damaged set **D** to intersect with **$BWS_i$** of each subgraph.
- Mark subgraphs as damaged if there's an intersection.

**Evaluate Data Items**

- For each damaged subgraph:
  - Check dependencies in CDS.
  - Compare last update times with attack time ($t_a$) and affected time ($t_{aff}$).

**Determine Damage Status**

- Classify items as damaged if:
  - Last update time equals affected time.
  - Item depends on a damaged parent.
- Release items if they are not part of the damaged lineage or have been recovered.

# Damage Assessment(Example)

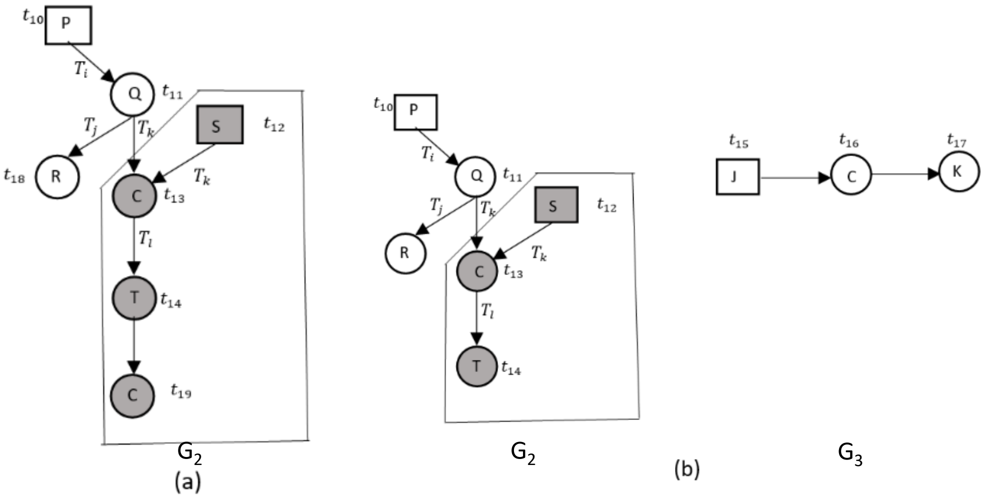



Figure 5: Multiple subgraphs in the data dependency (G).

TABLE 1: FINAL UPDATED TIMETABLE (FOR SCENARIO (A))

| Data Items | P | Q | S | T | C | R |
|---|---|---|---|---|---|---|
| $t_{\text{Last Updated}}$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{14}$ | $t_{19}$ | $t_{18}$ |
| Graph | $G_2$ | $G_2$ | $G_2$ | $G_2$ | $G_2$ | $G_2$ |

TABLE 2: FINAL UPDATED TIMETABLE (FOR SCENARIO (B))

| Data Items | P | Q | S | T | J | C | K | R |
|---|---|---|---|---|---|---|---|---|
| $t_{\text{Last Updated}}$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ | $t_{17}$ | $t_{18}$ |
| Graph | $G_2$ | $G_2$ | $G_2$ | $G_2$ | $G_3$ | $G_3$ | $G_3$ | $G_2$ |

## Scenario (a):

- Table 1 shows that data item ***C*** was last updated at $\boldsymbol{t_{19}}$.
- This update occurred in the damaged graph $G_2$ (shaded part).
- Since ***C*** is updated within the same damaged graph, it remains compromised.

## Scenario (b):

- Table 2 shows that data item ***C*** was last updated at $\boldsymbol{t_{16}}$ .
- This update occurred in a separate graph $G_3$.
- Although ***C*** is a child of the initially damaged data item ***S***, its update in a different graph signifies it is safe for release.

# Simulation Setup

**Objectives:**

Evaluate the efficiency and effectiveness of the Blind Write Lineage model in damage assessment.

**Variables Considered:**

1. Number of Transactions: 200 to 900
2. Number of Data Items: 500 to 3000
3. Max Operations per Transaction: 3 to 12
4. Max Write Operations: 1 to 5
5. Number of Blind Writes: 1% to 10% of transactions

# Simulation Results

### *Varying the number of transactions*

- As the number of transactions increases, the average data item reads in traditional logs rise gradually.
- This increase is due to more transactions leading to a higher number of blind writes and more subgraphs.
- The average data item reads remain relatively constant and significantly lower compared to traditional methods.
- This stability is attributed to consistent average dependency per graph, even with more transactions.

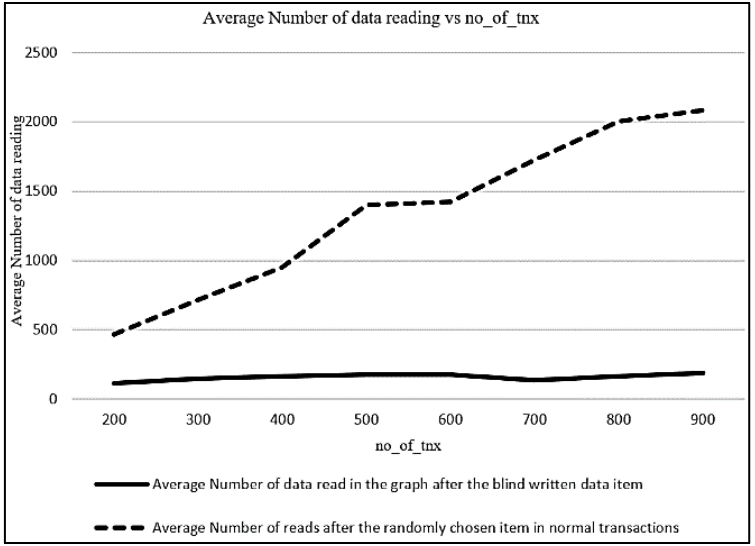



Figure 6: Varying the number of transactions.

# Simulation Results

***Varying the number of data items.***

- Significant decrease in average data reads after identifying damaged data using our method compared to traditional methods.
- The graph remains relatively consistent despite variations in the number of data items.
- This consistency is due to the fixed number of blind-written data items and written data items per transaction. Previously written items are often read later to write new items, leading to consistent behavior.

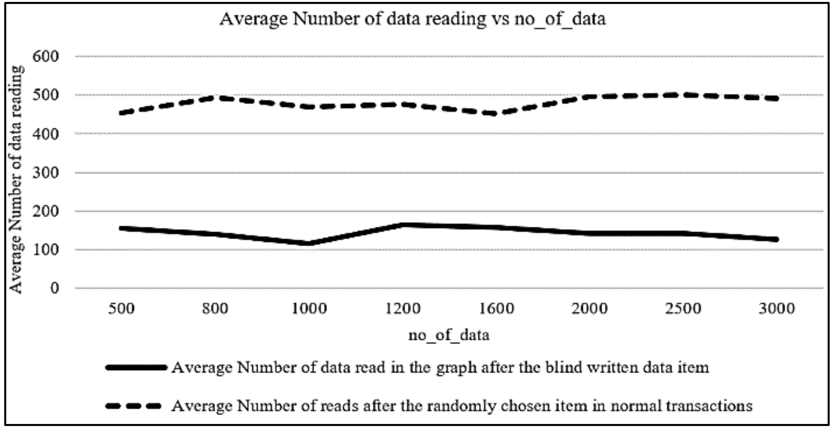



Figure 7: Varying the number of data items.

# Simulation Results

***Varying the Max number of operations per transaction.***

- Both methods show an increase, but our method maintains significantly lower average reads compared to traditional transactions.
- More operations per transaction lead to more read items. Increased dependency results in more data to read.
- Despite the gradual increase in reads, our method remains efficient, highlighting its effectiveness in managing dependencies even with higher operation counts. This explains the gradual increase observed in the graph.

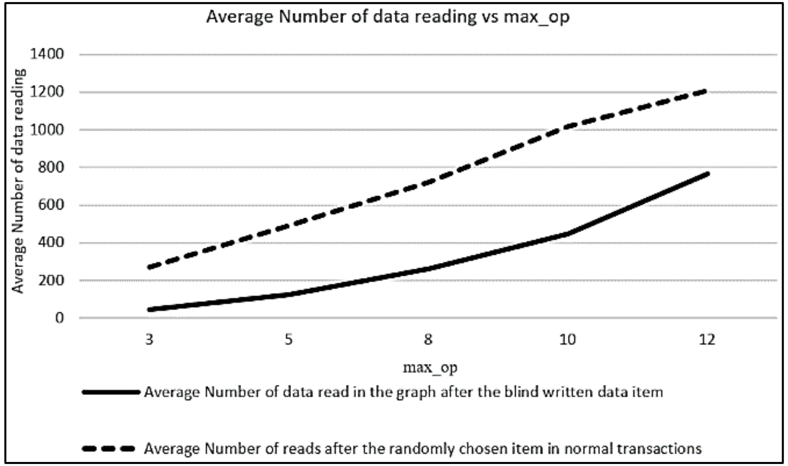



Figure 8: Varying the Max number of operations per transaction.

# Simulation Results

***Varying the Number of blind write per transaction***

- Our method shows a gradual decrease in average data reads. In contrast, normal transactions maintain relatively constant reads.
- In our method, as the number of blind writes increases, the number of subgraphs also increases. Consequently, the number of data items depending on each subgraph decreases, leading to a decrease in the average reading.

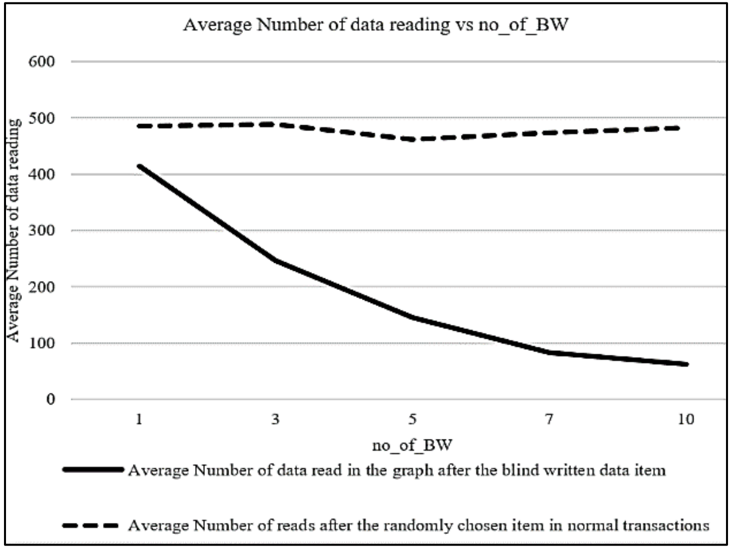



Figure 9: Varying the Number of blind write per transaction.

# Simulation Results

***Varying the Max write operations.***

- In traditional method, average data reads remain relatively constant.
- In our method, gradual increase in average data reads can be seen because more write operations lead to increased dependency.
- Fixed blind writes mean more data items are written after being read, increasing dependencies.

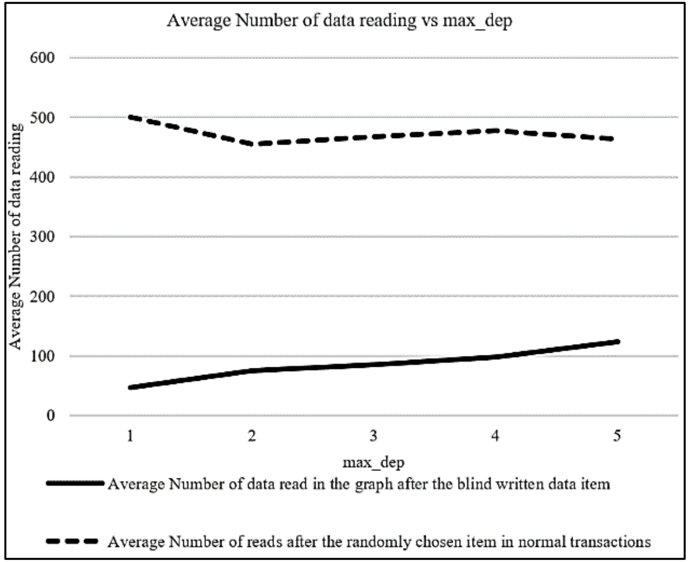



Figure 10: Varying the Max write operations.

# Conclusion

- Introduces an efficient technique for rapid damage assessment in fog computing systems
- Addresses limitations of traditional log analysis methods
- Leverages blind write lineage for efficient damage tracing
- Performance Advantages:
    - Superior speed in damage assessment
    - Enhanced efficiency in data recovery
    - Improved accuracy compared to traditional methods
- Future Work:
    - Refine model for specific time-range attacks.
    - Optimize memory usage with efficient data structures.
    - Ensure scalability across diverse architectures.
    - Explore blockchain for secure transaction logging.

UNIVERSITY OF ARKANSAS

# Thank You