Technische Universität Berlin

Chair of Database Systems and Information Management

Bachelor's Thesis

Creation of an In-network Pattern Evaluation Plan for Dynamic Fog-Cloud Environments

Nur Ali Dilek
Degree Program: Business Informatics
Matriculation Number: 456762

Reviewers
Prof. Dr. Volker Markl
Prof. Dr. Odej Kao

Advisor
Ariane Ziehn

Submission Date
12.08.2025

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 12.08.2025

. . . .

Nur Ali, Dilek

Zusammenfassung

Das rasante Wachstum von IoT-Geräten erzeugt massive und kontinuierliche Datenströme, was zu erheblichen Herausforderungen wie hohen Übertragungskosten und geringem Durchsatz für CEP in verteilten Umgebungen führt. Unter den beiden in CEP verwendeten Dezentralisierungsstrategien bestimmt die *Operatorplatzierung*, wo Operatoren im Netzwerk ausgeführt werden. Zweitens definiert die *Push-Pull-Kommunikation*, wie Ereignisse zwischen Knoten ausgetauscht werden. Um diese Herausforderungen zu bewältigen, kombiniert das bestehende INES-Framework diese Strategien und wendet eine Single-Node-Operator-Platzierungsstrategie (SNP) mit Push-Pull-Kommunikation (PrePP) in Fog-Cloud Umgebungen an. Dieser Ansatz ist jedoch auf eine Platzierung pro Unterabfrage beschränkt und weist in groß angelegten Topologien lange Rechenzeiten auf.

Aus diesem Grund erweitert diese Arbeit INES durch die Einführung von **INES***, wobei das SNP von INES durch eine Multi-Node-Platzierungsstrategie (MNP) mit erweiterter topologie- und ressourcenbewusster Platzierungslogik, sowie einer einfachen Graphkomprimierungstechnik ersetzt wird. MNP ermöglicht die Platzierung mehrerer Operatoren einer Unterabfrage über Fog- und Cloud-Schichten hinweg, wodurch die Netzwerkauslastung reduziert wird, während die Graphkomprimierung die Berechnungszeit für die Platzierung durch das Filtern irrelevanter Knoten verkürzt. Die Bewertung von Topologien mit bis zu 300 Knoten zeigt, dass unser Ansatz die Übertragungskosten im Vergleich zu SNP um bis zu **37%** reduziert, wobei die Übertragungsverzögerungen um **20-35%** zunehmen. Die Reduzierung der Übertragungskosten fiel geringer aus als erwartet, da die Architektur unserer Simulationsumgebung nicht auf den MuSe-Ansatz zugeschnitten ist. Das bedeutet, dass nicht alle Knoten miteinander verbunden und nicht bidirektional sind, da wir eine hierarchische DAG-Struktur haben, in der alle Eingangsströme nach oben verlaufen. Unsere Graphkomprimierung verbessert die Laufzeit um bis zu **14%** in großem Maßstab mit 300 Knoten ohne zusätzliche Übertragungskosten. Diese Ergebnisse zeigen, dass INES* eine skalierbare und anpassungsfähige CEP-Lösung für dynamische Fog-Cloud-Umgebungen ist, mit klarem Potenzial für weitere Optimierungen im Bereich Latenzbehandlung und reale Einsätze.

Abstract

The rapid growth of IoT devices generates massive and continuous data streams, creating significant challenges such as high transmission costs and low throughput for CEP in distributed environments. Among the two decentralization strategies used in CEP, *operator placement* determines where operators are executed in the network. Second, *push-pull communication*, which defines how events are exchanged between nodes. To overcome these challenges, the existing INES framework combines these strategies and applies a single-node operator placement (SNP) strategy with push-pull communication (PrePP) in fog-cloud environments. However, this approach is limited to one placement per subquery and exhibits long computation times in large-scale topologies.

For this reason, this thesis extends INES by introducing **INES***, where INES's SNP is replaced with a multi-node placement (MNP) strategy with extended topology- and resource-aware placement logic as well as a simple graph compression technique. MNP allows multiple operator placements of one subquery across fog and cloud layers, reducing network load, while graph compression shortens placement computation time by filtering irrelevant nodes. The evaluation on topologies of up to 300 nodes shows that our approach reduces transmission costs by up to **37%** compared to SNP, with a **20-35%** increase in transmission delays. The reduction in transmission costs has been less than expected, as the architecture of our simulation environment is not tailored to the MuSe Approach. This means that not all nodes are linked to each other and are not bidirectional, as we have a hierarchical DAG-structure in which all input streams go upwards. Our graph compression improves runtime by up to **14%** in large-scale of 300 nodes without additional transmission costs. These results demonstrate that INES* is a scalable and adaptable CEP solution for dynamic fog-cloud environments, with clear potential for further optimization in latency handling and real-world deployments.

Contents

List of Figures

List of Tables

List of Abbreviations

DIMA	Database Systems and Information Management
MuSE	Multi-Sink Evaluation
INEv	In-Network Evaluation
PrePP	Predicate Based Push-Pull
CEP	Complex Event Processing
INES	In-Network Evaluation for NebulaStream
INES*	In-Network Evaluation for NebulaStream with this work's extension
IoT	Internet of Things
DAG	Directed Acyclic Graph
SNP	Single Node Placement
SNP_{GC}	Single Node Placement with computational complexity
MNP	Multi Node Placement
MNP_{GC}	Single Node Placement with computational complexity
OP	Operator Placement
PP	Push-Pull

List of Algorithms

1 Introduction

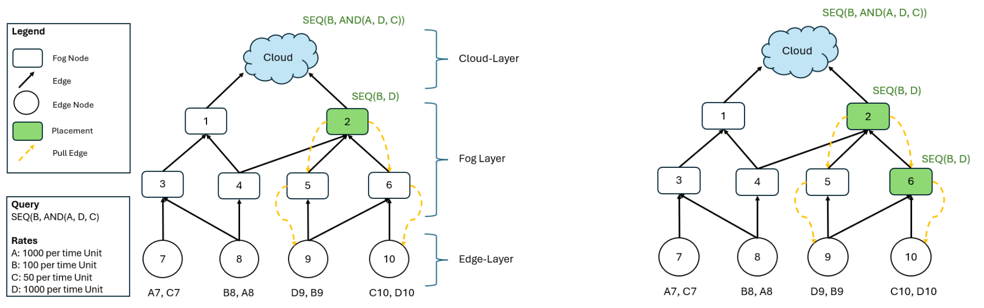
The rapid growth in the usage and number of Internet of Things (IoT) devices has led to an unprecedented global increase in data volume [30]. This rapid expansion of IoT devices and data volume presents significant challenges for traditional cloud-based environments, including high transmission costs, low throughput, and increased delays due to longer data paths and centralized processing bottlenecks [30]. To address these issues, unified fog-cloud environments have been proposed, where fog nodes act as an extension of the cloud. These unified environments enable stream processing systems (SPSs) to leverage the resources attached to the fog and tailor data close to its source while using the cloud as a central fallback. As a result, utilizing the fog resources enables SPSs to reduce transmission costs and enhance resource efficiency, crucial for the IoT.

Complex Event Processing (CEP) is a stream processing paradigm that detects complex patterns in event streams created, e.g. by IoT devices and triggers actions based on pattern matches [15]. These properties make CEP an essential tool for analyzing and reacting promptly to the ever-growing amount of data resulting from the expansion of the IoT, with application domains spanning from financial services to emerging healthcare systems and sensor networks [5, 7, 11, 17]. Relying solely on centralized processing becomes a bottleneck, for these applications prompting the adoption of unified fog-cloud environments. In this context, enabling efficient CEP requires adapting traditional decentralization strategies to the constraints of hierarchical topologies and the heterogeneous hardware landscape of fog-cloud environments, especially under demanding IoT workloads. Two primary strategies are commonly applied to decentralize streaming workloads:

(1) **Operator placement (OP)** decomposes queries into sub-operators and strategically assigns them to nodes in the network to exploit locality and available compute resources [5]. Intermediate results produced by these sub-operators need to be exchanged across nodes to compute the final query result. Placement decisions are typically guided by cost functions, most commonly minimizing upstream data movement, i.e., transmission costs between nodes.

(2) **Push-pull communication (PPC)** reduces transmission costs by pushing low-volume streams and pulling high-volume ones on demand from descendant nodes [10, 3, 25].

INES (In-Network Evaluation for NebulaStream) [21] integrates both decentralization strategies into a unified framework. Figure 1a illustrates how INES



(a) Single-Node Placement. (b) Multi-Node Placement.

Figure 1: Motivating Example.

operates using a representative complex pattern, `SEQ(B, AND(A, D, C))`, composed of four distinct event types (i.e., logical streams). The depicted topology consists of 11 nodes structured across three layers: a cloud layer (1 node), a fog layer (6 nodes), and an edge layer (4 nodes). In the edge layer, source nodes emit substreams of various event types, e.g., source node $\textcircled{7}$ produces events of types **A** and **B**. Each event type generates events (also referred to as tuples) at different rates; for example, event type **A** emits 1000 tuples per time unit. By decomposing the pattern, INES is able to extract and push down the subquery `SEQ(B, D)` to node $\boxed{2}$, reducing upstream transmission costs. Additionally, INES applies push-pull communication to optimize data exchange further: node $\boxed{2}$ receives low-volume streams **B** via push and selectively pulls high-volume streams **D** only when needed, minimizing unnecessary data transfer further. In particular, depending on the selectivity of the subquery (i.e., how many of the pulled events are actually needed for matches), INES is able to reduce the transmission costs of up to 90% [21]. However, when a subquery is placed entirely on a single node in the topology, this is referred to as single-node placement (SNP), which often leads to placements high in the hierarchy, close to the cloud, where all substreams of the involved event types converge [21]. While this simplifies coordination and computation of the NP-hard problem of OP [15, 5], it leaves many lower-layer resources underutilized, limiting scalability and transmission cost savings. To address this limitation, Multi-Node Placement (MNP)[5] has been proposed. Unlike SNP, MNP distributes subqueries across multiple nodes, allowing partial evaluation wherever a single event type converges at a node closer to data sources. As contrasted in Figure 1a and Figure 1b, MNP enables the placement of parts of the subquery `SEQ(B, D)` on different nodes, e.g. nodes $\boxed{2}$ and $\boxed{6}$. This increased flexibility reduces upstream data movement and increases fog-layer utilization. To overcome the limitations of INES and fully exploit fog-layer resources, we propose INES*, an extension of INES that supports MNP in combination with PPC.

Challenges and Research Questions. This extension introduces several challenges. First, integrating MNP into INES is non-trivial, as it requires rethinking how operators are distributed across the network while balancing transmission costs with resource utilization and latency. Second, the existing INES approach already incurs high computation time due to the combinatorial complexity of OP and PPC, e.g., generating an execution plan for a topology with 1000 nodes using INES takes up to three hours. Allowing MNP further expands the execution plan search space exponentially, increasing the planning time substantially [5, 21]. To address these challenges, this thesis investigates the following research questions (RQ):

- RQ1: How can MNP be efficiently integrated into INES?
- RQ2: What is the impact of MNP on transmission cost compared to INES with MNP?
- RQ3: What are the key contributors to INES's computation time, and how can execution plan generation be accelerated to make MNP feasible in practice?

Structure of the thesis. The remainder of this thesis is organized as follows: Chapter 2 presents relevant background, including the environmental architecture, operator placement, push-pull communication, and graph compression techniques. Next in in Chapter 3, we illustrate our research problem by analyzing the architecture of the INES framework and identifying integration points for our extensions. Chapter 4 details the design and implementation of multi-node placement and graph compression within INES*. Afterwards, in Chapter 5, we present the evaluation methodology and experimental results, focusing on the computation time, transmission cost and latency. After evaluation our analysis, we present related work (Chapter 6). Finally, Chapter 7 summarizes our contributions, discusses limitations, and outlines directions for future work.

Anticipated Impact

Efficient operator placement for CEP in fog-cloud environments has significant real-world relevance [5, 21]. It enables timely event processing close to the data sources, which is critical for applications like smart cities, industrial monitoring, and healthcare [7, 8, 9]. By reducing transmission, INES* may also be qualified to serve as a baseline for future research in real-world systems such as NebulaStream (NES). It may also contribute to scalable, energy-efficient and responsive IoT infrastructures that support data-intensive and latency-sensitive applications [11, 17].

2 Scientific Background

This chapter provides foundational background information relevant to our work. Section 2.1 introduces different network environments. Following with an overview about Complex Event Processing (CEP), outlining the preliminaries, including the network model and the query language used throughout this work (Section 2.2). Next, Section 2.3 presents two approaches of the CEP distribution strategy, operator placement, and push-pull communication. Finally, Section 2.4 offers different graph compression techniques.

2.1 Network Environments

Network environments can be represented using directed graph models, where nodes N denote computing or sensing devices and directed edges E represent communication links between them [20]. These models form the foundation for analyzing data processing strategies across distributed computing layers.

In this section, we differentiate between three environments the *cloud environment*, *fog environment* and the combination of both environments, the *fog-cloud environment*. Thereby, we compare their characteristics, advantages, and limitations. The fog-cloud environment, which combines elements of both, represents the operational simulation environment and introduces benefits as well as unique challenges for operator placement.

2.1.1 Cloud and Fog Models

From a network perspective, *cloud environments* are typically modeled using a star topology [1]. In this structure, multiple edge nodes are directly connected to a centralized cloud node, which serves as the central point for all communication and computation. Direct links between edge nodes themselves are non-existent. This means that all communication and data aggregation must pass through the central cloud node [2, 8, 30].

The *fog environment* is typically represented as an undirected mesh graph comprising decentralized nodes without a central controlling unit [16]. This topology allows flexible communication patterns among fog nodes, facilitating local collaboration and partial query evaluation. In contrast to the star-topology structure of cloud environments, fog nodes can exchange information directly with each other and spare the communication path via one central unit [30, 31].

2.1.2 Fog-Cloud Environment

The fog-cloud environment integrates both cloud and fog layers into a hierarchical, directed acyclic graph (DAG) topology [2]. It typically consists of:

- An **edge layer** with data-generating nodes (e.g., sensors)
- One or more **fog layers** with intermediate compute nodes
- A centralized node in the **cloud layer**.

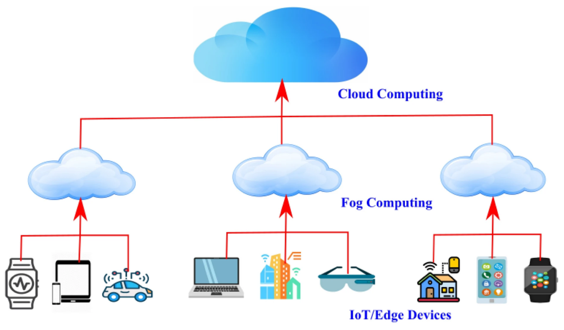


Figure 2: Fog-Cloud Computing Model.
Source: [2]

Figure 2 visualizes the architecture of the fog-cloud model, with data-generating **Edge/IoT nodes** at the bottom and transmitting data to nearby **fog nodes**. These fog nodes serve as intermediaries, processing data closer to its source. Finally, relevant information is transmitted to the central **cloud**.

Comparison of Network Characteristics. The Table 1 highlights the characteristics of each network environment and contextualizes the benefits of combining the two environments cloud and fog int a fog-cloud environment:

Attribute	**Cloud**	**Fog**	**Fog-Cloud**
Network distribution	Centralized	Decentralized	Hybrid
Latency	High	Low	Mixed
Processing power	Scalable	Limited	Scalable
Storage capacity	High	Limited	High
Topology	Star	Mesh	Hierarchical DAG

Table 1: Comparison of cloud, fog, and fog-cloud network environments

As illustrated in Table 1, **cloud environments** offer virtually unlimited computing and storage capacity but often suffer from high latency due to centralized processing [30]. In contrast, **fog environments** enable low-latency, localized processing close to the data source but are constrained by limited computational and storage resources [6, 23].

The **fog-cloud environment** combines the strengths of both: scalable processing and storage capabilities from the cloud with low-latency, proximity-based processing from the fog. By distributing processing tasks across layers—handling data in the fog layer whenever possible and forwarding only aggregated or relevant results to the cloud. This hybrid model can significantly improve scalability, efficiency, and responsiveness [2, 8, 9].

However, this hybrid structure introduces additional complexity when aiming for efficient operator placement: multiple nodes across different layers may serve as potential placement candidates, and the computational load must be carefully balanced between fog and cloud resources, considering the heterogeneous capabilities of the devices. [2].

2.2 Complex Event Processing

Complex Event Processing (CEP) is a stream processing paradigm that detects complex patterns in event streams created by IoT devices and triggers actions based on pattern matches [21, 31]. Thus, enabling efficient CEP in a fog-cloud environment is essential to support emerging large-scale IoT applications.

Preliminaries

This section fixes the **fog-cloud setting** used throughout the thesis. We build on the background in Section 2.1 and only formalize the operational *network model* and the *query language* assumed by our placement and routing algorithms.

Network Model

In this work, the network is modeled as a Directed Acyclic Graph (DAG) with N nodes, where nodes represent event-generating or processing devices, and edges represent unidirectional communication paths between them.

The structure follows a hierarchical topology, in which nodes are organized in layers:

- **Source nodes** are located in the lowest layer and are responsible for generating raw event data. These nodes have no computing capabilities.

- **Fog nodes** are placed in intermediate layers and have limited computational and storage resources. Their capabilities typically increase with their position in the hierarchy.

- A **cloud node** in the top layer, representing a centralized system with virtually unlimited computational power and storage capacity.

Each child node can be connected to one or more parent nodes. Communication is primarily upward in the hierarchy. However, pull requests for event data may also be initiated from parent nodes toward their children, enabling bi-directional information flow for certain communication models. This hierarchical and heterogeneous model captures the resource diversity and structural characteristics of fog-cloud environments, making it well-suited for evaluating event processing strategies across different layers of computation and data transfer capabilities [6, 13].

Query Language

For CEP, a common query language is adopted. A query is composed of operators, a set of predicates that define conditions based on the payload data of events, and a time window [6]. The query is represented as an ordered tree structure, where internal nodes correspond to composite operators and leaf nodes correspond to primitive operators:

- **Primitive Operators:** Detects a specific type of event and has no child operators.

- **Composite Operators:** Have child operators and detect based on their results event patterns.

In our work, we focus on two composite operator types:

- **AND:** Identifies a pattern if a track contains patterns as defined by all its child operators, in any order.

- **SEQ:** It is similarly to **AND** but additionally consider the order of the events.

Query Decomposition. To improve efficiency, as queries have become more complex, they will be split into sub-queries and evaluated at the assigned network nodes. This decomposition process, known as projection, involves breaking down the original query into subqueries. Each projection targets a specific subset of event types while maintaining the predicates and time constraints of the original query. To preserve the correctness of the evaluation, these projections are organized as a vertex into a directed acyclic graph (DAG) structure. Approaches such as MuSE [6] and INEv [5] employ this query splitting strategy to optimize operator placement within distributed CEP environments.

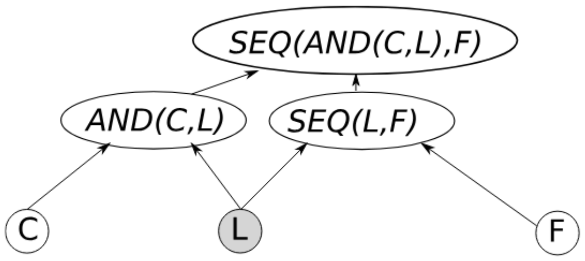


Figure 3: Query Splitting.
Source: [6]

Figure 3 illustrates an example of query decomposition. The original query `SEQ(AND(C, L), F)` is split into two projections: `AND(C, L)` and `SEQ(L, F)`. The overlapping event type **L** appears in both projections. This ensures semantic correctness and enables each projection to be processed closer to the relevant data sources [5, 6].

By processing smaller sub-queries early in the network (e.g., at fog nodes), the system can filter irrelevant data before it reaches more centralized resources. This selective filtering reduces overall data transmission and improves latency — especially when the projections have low selectivity, meaning they discard many events.

Placement Strategy. For the evaluation, the projections of the query are assigned to network nodes. The assigning can occur in two different ways, as mentioned in Section 2.3.1 SNP and MNP:

- **Single-Node Placement:** Here all events in the network are processed in a single Node.
- **Multi-Node Placement:** Distributing a query by processing events at multiple Nodes in the network.

Despite the placement strategy, it is important to note that splitting a query into sub-queries in an optimal manner is **NP-hard** [5, 6, 15].

2.3 Distribution Strategies of CEP

In this section, we present two decentralization strategies of CEP: operator placement with the SNP and MNP strategy and push-pull communication models, which are crucial to manage massive data streams in distributed environments.

2.3.1 Operator Placement

Operator placement plays a crucial role in efficiently evaluating CEP queries within distributed networks. In traditional CEP systems, query evaluation takes place at a centralized sink node. However, this approach can lead to bottlenecks, high latency, and inefficient resource utilization, especially on a large scale.

To overcome these limitations, distributed CEP frameworks leverage operator placement strategies, which assign the operators of a query to multiple nodes in the network. This decentralized query execution enables parallel processing, reduces communication overhead, and improves scalability and responsiveness [11, 25]. Additionally, placing operators closer to data sources can significantly reduce latency and network bandwidth consumption, leading to better performance and efficiency [5].

However, despite its benefits, the operator placement problem is computationally challenging. It is known to be **NP-hard**, as it requires optimizing multiple conflicting objectives, simultaneously [5, 15, 21]. These objectives may include minimizing total communication and transmission costs, respecting resource constraints of heterogeneous nodes such as CPU and memory, reducing latency through proximity-aware placement, and balancing load to prevent communication overheads. These conflicting goals are typically captured in a cost function, which the placement algorithm seeks to minimize [5, 6, 28].

To address the complexity of this problem, current research proposes various heuristics, approximation algorithms, and cost-based optimization techniques [5, 6]. These include latency-aware placement, resource-aware scheduling, and hybrid strategies tailored to fog-cloud infrastructures [21, 28]. These strategies are essential for enabling scalable and efficient CEP in distributed environments such as a fog-cloud environment. In the context of this thesis, operator placement is a central component. We build upon the INES framework [21], which already employs the cost-based state-of-the-art approach INEv with single-node placement [21]. Our contribution extends this work by introducing *multi-node placement* into INES, enabling projections to be evaluated in parallel across multiple locations in the network.

State-of-the-art Approach. The *In-Network Evaluation* (INEv) model [5] in the INES framework addresses the challenge of reducing communication overhead in distributed CEP. Unlike centralized approaches where all raw events are transmitted to a cloud node for processing, INEv splits a query workload into sub-queries, placing them at network nodes, and forwards their results upstream to other nodes before reaching the central node [5]. This strategy minimizes unnecessary transmissions, particularly for selective queries with high input rates. A key aspect of INEv is its cost-based optimization framework. Given a network topology and the statistical characteristics of incoming event streams (such as event rates and selectivities), it searches for an efficient operator placement plan [5, 21]. The placement is restricted to single-node assignment, meaning that each projection is evaluated at exactly one node. This simplifies coordination but may lead to suboptimal use of available resources. In the context of INES, INEv serves as the foundational mechanism for operator placement. INES adopts its query decomposition and placement strategy, but extends it by integrating additional models like the Predicate-Based Push-Pull (PrePP) model, enabling even greater flexibility and transmission reduction [21].

Operator Placement Strategy. Different operator placement strategies are employed in distributed environments, often in combination with evaluation at a single sink node and assigning sub-queries only once within the network topology [5, 21]. This subsection contrasts **Single-Node Placement** and **Multi-Node Placement**, discussing their respective benefits and limitation to highlight the importance and potential of multi-node placement strategies in dynamic fog-cloud environments, where scalability, latency, and efficient resource utilization are critical.

Single-Node Placement. In CEP, Operator placement has traditionally a single-sink node, in which all query results are stored [4]. With the sn-placement strategy, we are able to split the query into a subquery and place it one time in the topology [4, 6, 21].

Benefits. This placement strategy simplifies result collection and reduces coordination overhead. Additionally, it enables centralized optimization of result storage and aggregation [6, 5, 21].

Limitations. Despite the benefits of single-node placement, it inherently limits its flexibility and scalability. Current systems typically place each sub-query or operator only once in the network, without considering the potential advantages of duplicating certain operators and placing them on multiple nodes [4, 6]. The placement-strategy could potentially lead to a centralized bottleneck that can affect the scalability and overall performance, such as latency and high data transmission, by high event rates or in large-scale networks [4, 6, 24].

Multi-Node Placement. In contrast to single-node placement, multi-node placement allows operators to be processed on multiple nodes, potentially resulting in multiple placements for a given query [4]. This way, the placement strategy addresses the limitations of the single-node placement. The goal is to reduce data transmission overhead and alleviate central bottlenecks by executing operators closer to event sources. The decision as to whether a projection is executed several times is based on cost models that include network transmissions and selection effects as follows [5]:

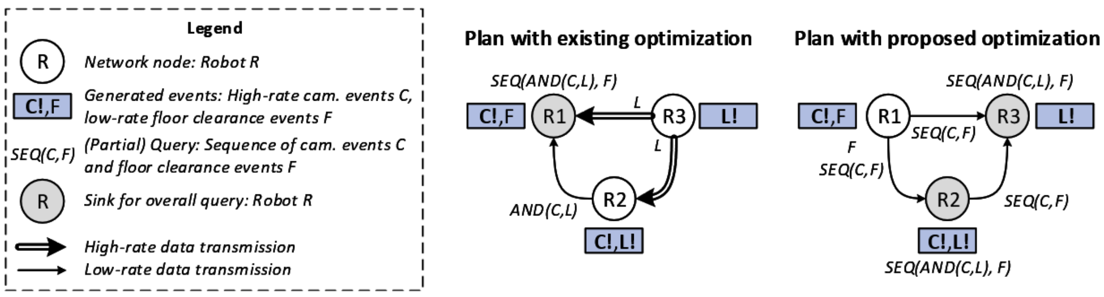

Legend
R Network node: Robot R
C!,F Generated events: High-rate cam. events C, low-rate floor clearance events F
SEQ(C,F) (Partial) Query: Sequence of cam. events C and floor clearance events F
R Sink for overall query: Robot R
High-rate data transmission
Low-rate data transmission
Plan with existing optimization
SEQ(AND(C,L), F)
C!,F
R1
L
R3
L!
L
AND(C,L)
R2
C!,L!
Plan with proposed optimization
SEQ(AND(C,L), F)
C!,F
R1
R3
L!
SEQ(C,F)
F
SEQ(C,F)
R2
SEQ(C,F)
C!,L!
SEQ(AND(C,L), F)

Figure 4: Multi-Node Plan.
Source: [6]

The following figure (Figure 4) shows the benefits of multi-node placement by comparing it with the single-node placement plan:

- **Robot 1**: Generates **event C** at a high rate and **event F** at a low rate.
- **Robot 2**: Generates **event C and L** at a high rate.
- **Robot 3**: Generates **event L** at a high rate.

Single-Node Plan. To reduce high-rate data transmission, the *plan with existing optimization* is introduced, showing *single-node placement strategy*: With the optimized operator placement, the query is decomposed into sub-operators, for example, `AND(C,L)`, which is more selective than sending raw event streams. This optimization results in lower data transmission rate and, consequently, reduced transmission costs. However, despite the use of this plan, data still needs to be routed through a central node, which can remain a potential bottleneck [6].

Multi-Node Plan. The *plan with the proposed optimization* is showing an example of the *multi-node placement strategy*. It enables a strong reduction in transmission costs compared to the single-node models by using arbitrary query projections (i.e., `SEQ(C,F)`) and multiple sinks, such as Robot ② and Robot ③. This eliminates the central bottleneck, as different parts of the request are processed in parallel and a projection is placed multiple times in the network [6].

Benefits. The multi-node placement strategy offers several benefits. Most notably, it reduces both transmission costs and latency by distributing query processing across multiple nodes in the network topology [4, 5, 6]. This decentralization enables a more scalable architecture that avoids the limitations of central processing. Furthermore, it improves resource utilization and fault tolerance [24, 21, 4].

Limitations. While multi-node placement offers different benefits, it also introduces limitations. The placement strategy increases the complexity in result merging and consistency handling with increased distribution of operators. Furthermore, managing duplicate processing can lead to a higher coordination overhead. Finally, the cost-based placement strategy may require significant computation time, especially in large-scale networks [5].

2.3.2 Push-Pull Communication

Another crucial decentralized strategy is the push-pull communication strategy in optimizing data exchange between nodes. Push-pull leverages the edges of a network topology to push or pull data depending on network and workload conditions. The communication model consists of the two following fundamental components [3, 12, 14]:

- **Push Communication:** Here, the producer proactively sends data to a consumer as soon as it becomes available. This model is beneficial, when data is needed continuously or immediately [3, 14].
- **Pull Communication:** In contrast, the receiver sends a request to the producer. It is effective, when data is required infrequently or when only a part of high-frequent data is necessary [3, 14].

In CEP, distributed nodes produce event streams, and the CEP system analyzes these streams to identify significant patterns (composite events). In the traditional approach, CEP systems send all events to the centralized processing node, which is why typically a push-based model is used [12, 3]. This centralized push approach leads to high transmission and inefficiency in the network due to high event rates, in which only a subset of data is relevant [14]. An improved strategy is the hybrid push-pull communication model, addressing these issues by selectively transmitting data along the nodes. The hybrid approach leverages event rate characteristics and temporal relevance to reduce unnecessary data transfers and improve overall communication efficiency [3, 14].

Push-Pull Models.

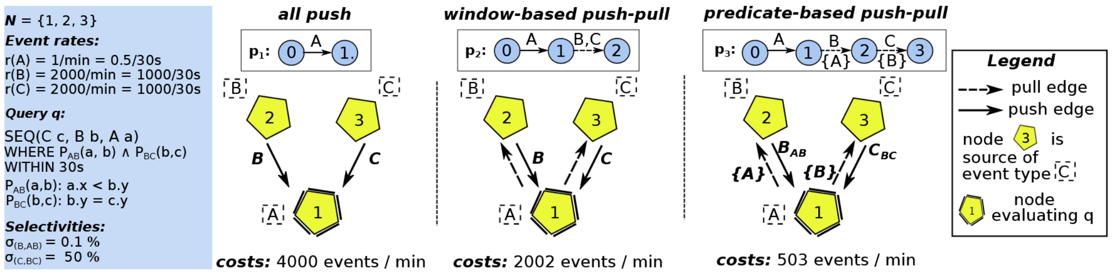


Figure 5: Push-pull Communication Models.
Source: [25]

Events in CEP systems are usually time-sensitive and exhibit irregular distribution patterns across the nodes. To ensure the timely detection while minimizing network overhead, PPC strategies must be adapted to the frequency of events and the time constraints. To illustrate these strategies, Figure 5 presents three communication models along with their respective impacts on transmission costs.

All Push. In the *all-push communication model* source nodes forward the events immediately to the evaluating node 1. As a result, the communication model has the most transmissions costs, reaching up to **4000 transmitted events per minute** [25].

Window-Based Push-Pull. Compared to the *all-push communication model*, we reach with the *window-based push-pull communication model* a reduction of 50 percent with up to **2000 transmitted events per minute**. This is due to the fact that the evaluation node sends pull requests to source nodes 2 and 3, which then respond by pushing only those events that fall within the relevant time window constraints [25].

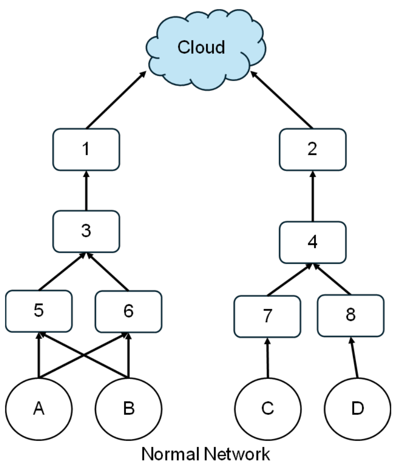
State-Of-The-Art Approach: Predicate-Based Push-Pull Finally, the *predicate-based push-pull (PrePP)* model represents the state-of-the-art approach of push-pull and is implemented in **INES**. In this model, source nodes apply predicate-based filtering before transmitting events in response to pull requests from the evaluation node. Consequently, pull requests include not only the timestamp of the **A** event, but also its attribute values required to evaluate the predicates—specifically for matches between **A** and **B**, as well as **B** and **C**. This selective filtering leads to a further reduction in transmission costs by up to 75% compared to the window-based push-pull communication model by transmitting approximately **500 events per minute** [25].

2.4 Graph Compression

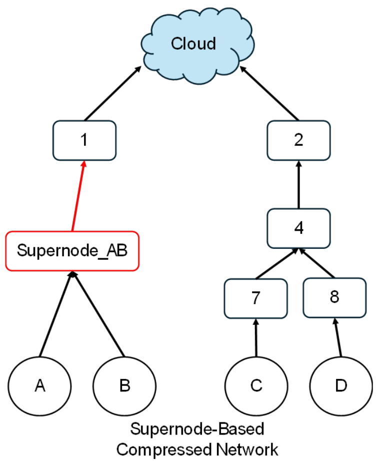
In CEP, the more nodes that exist, the more complex the placement of operators within the network becomes [21]. Finding an approximately optimal placement in a CEP network is associated with high computation time due to the fact, that OP is a NP-hard problem [5, 6, 21]. To reduce the computation time required for finding an execution plan, various graph compression techniques can be employed to decrease the effective problem size. In this section, we present 3 different Graph Compression algorithms: The *supernode-based graph compression* 2.4.1, which reduces the network size by aggregating nodes and edges. Next, the *Edge-grouping method*(2.4.2), which dedensifies the network and finally, the *simplified-based method*(2.4.3), which reduces the network size by removing less important nodes in the network.

2.4.1 Supernode-Based Graph Compression

The implementation of the **supernode-based graph compression** approach reduces the complexity of shortest-path computation and operator placement in large-scale CEP networks, by aggregating nodes into supernodes. The core idea is to group structurally or behaviorally similar nodes into a single supernode, thereby reducing the overall graph size without significantly affecting routing accuracy [29]. A **supernode** represents a cluster of original nodes with comparable roles in the network topology, for instance, edge devices transmitting the same event types and are connected to similar upstream nodes [29]. The so-called **superedges** are the edges between the supernodes, and summarize the connections between all corresponding nodes of the source and destination supernodes [29].



(a) Original Graph.



(b) Compressed Graph.

Figure 6: Example of the Supernode-Based Compression Technique.

The example illustrated in Figure 6 demonstrates the influence of applying the supernode-based compression technique to a CEP-network topology. The original topology 6a consists of 13 nodes: 4 source nodes, which generate the events **(A,B,C,D)**, 3 fog layers with 8 nodes, and on top is the cloud node. The events **A** and **B** are forwarded from the same source to node 5 and 6. Afterwards, both nodes send these events to node $\boxed{3}$. This redundancy is an ideal candidate for compression. In the compressed topology 6b, we merge nodes $\boxed{5}$ and $\boxed{6}$ into the **Supernode_AB**. The edge from the Supernode_AB to node $\boxed{1}$ demonstrates the superedge.

Observing Points While supernode-based graph compression is valuable for initial pruning or coarse placement strategies [26], it should be combined with validation on the uncompressed topology to ensure feasibility and performance, thereby maintaining the correctness of the placements in the topology (such as cost calculation, event coverage, and placement).

Benefits for CEP This approach, applied to CEP, brings the following benefits:

- *Redundant sources* (e.g., multiple nodes emitting the same primitive event) are grouped.
- Paths used for projection placement are computed on the compressed graph.
- The resulting placements are then mapped back to the original topology.

Through supernode summarization, this approach reduces the number of graph nodes and edges used during operator placement in the network. Thereby, the calculation time of both the placement logic and the required shortest-path computations should be reduced due to the reduced complexity [26].

2.4.2 Edge-Grouping Method

Graph dedensification is an edge-grouping compression method that reduces the number of edges by introducing so-called compressor nodes. This technique is particularly effective in graphs with high-degree nodes, where many edges may be redundant or structurally similar [19, 26]. Instead of merging nodes (as in supernode compression, see Section 2.4.1), dedensification identifies dense neighborhoods around high-degree nodes and replaces sets of edges with a single intermediary node (a compressor node). This node connects to the high-degree node and its neighbors, collapsing redundant connections into a single virtual structure [26].

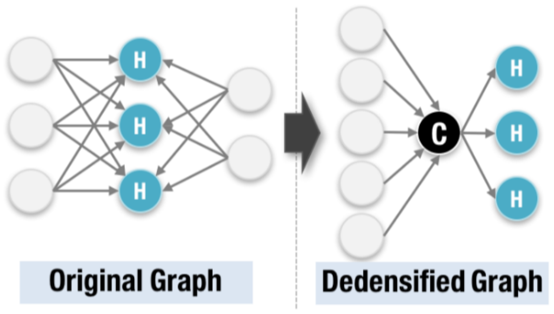


Figure 7: Edge-Grouping: Dedensifiying Graph.
[26]

Figure 7 illustrates the *dedensification method*. In the original graph, many nodes have multiple outgoing edges to high-degree nodes, creating dense neighborhoods. Through dedensification, each original node keeps at most one outgoing edge to a compressor node. Next, the incoming edges to the high-degree nodes are only received from compressor nodes. This approach enables pattern matching with query processing algorithms directly on the compressed graph [26].

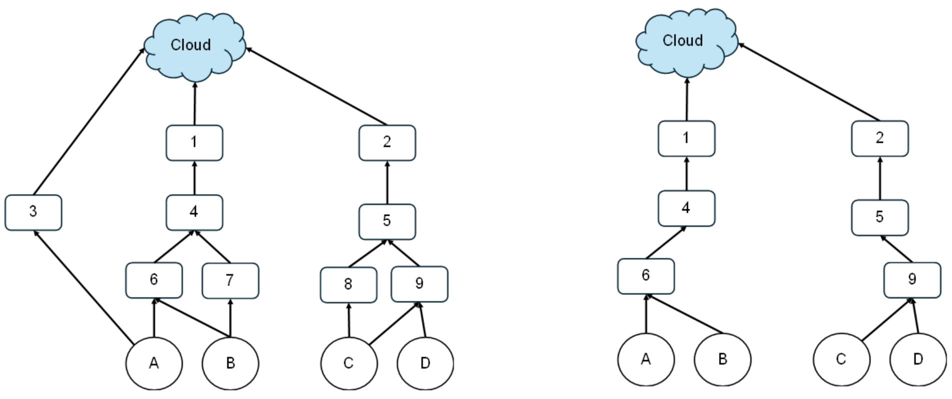
Observing Points While edge-grouping improves structural sparsity, it faces a major obstacle, the architectural structure of our environment. This is due to the fact that this technique include new elements, e.g. compressor nodes. These elements would require adapted query semantics and graph traversal logic to fit it into our framework.

Benefits for CEP This method reduces the edge complexity in areas with high connectivity. Furthermore, it retains lossless compression guarantees, ensuring query results remain exact and enhancing scalability for event pattern routing in dense subgraphs.

2.4.3 Simplification-Based Method

Simplification-based graph compression reduces the size and complexity of a graph by removing less important nodes or edges, resulting in a sparsified version of the original graph [18, 27]. In contrast to node-grouping or edge-grouping methods, this approach keeps a subset of the original graph and aims to preserve essential structural properties [26].

A representative example is OntoVis [27], which performs node filtering based on semantic and structural abstraction. Nodes are removed based on their degree, type, or importance, allowing users to focus on relevant substructures [26].



(a) Original Graph. (b) Simplified Graph.

Figure 8: Example of the Simplification-Based Technique.

The following figure 8 shows an example implementation of the simplified-based technique in our CEP-network topology. The `original` topology, illustrated in figure 8a, consists of 13 nodes: 4 event-generating source nodes, which generate the events **(A,B,C,D)**, 3 fog layers with 8 nodes, and on top is the cloud node. We reduce the size of our network by filtering fog nodes, which contain less than 2 events, out of the network, as a placement requires a minimum of two events per node. As a result, we are able to reduce the network size by 3 nodes (see Figure 8b).

Observing Points However, when applying this simplification-based approach, we we must observe the correctness of the costs and hops calculations, as nodes and paths are removed. Additionally, we must ensure that all event type bindings (ETBs) are still processed correctly and that no relevant nodes are excluded from placement due to over-filtering.

Benefits for CEP The implementation of this method has several advantages for our CEP-Network:

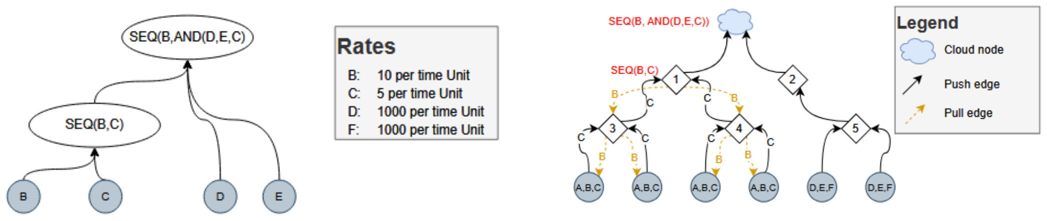
- It removes redundant or low-impact nodes for faster operator placement.
- Enables pruning of parts of the graph that are unlikely to host computation.
- Reduces memory and time complexity during shortest-path calculations.

While simplification may result in loss of global structure, it is highly effective for initial heuristics, interactive visualization, or approximate placement when full precision is not required [26].

3 Research Problem

We use for our work the INES Framework as baseline [21]. INES is an in-network Evaluation for NebulaStream (INES) introduced by Mr. Meran [21] and combines the state-of-the-art approaches of the two named CEP-strategies *In-Network Evaluation* (INEv) [5] and *predicate-based push-pull* (PrePP) [25] in a hierarchical fog-cloud environment [21]. The hierarchical structure consists of multiple layers starting with the edge layer, along with one or more fog layers to the centralized cloud server with heterogeneous edge and fog nodes in the corresponding layers [21].

The following Figure 9 illustrates the sequential combined communication model. First occurs the query splitting mechanism 9a and afterwards the push-pull communication 9b:



(a) Operator Tree.   (b) Network Placement.

Figure 9: INES Communication Model.
[21]

The Query `SEQ(B, AND(D, E, C))` will be splitted into the subquery `SEQ(B, C)` (see 9a), which is placed and evaluated at node one, as it covers all event bindings of the events **B** and **C** (see subfigure 9b). INES then applies the PrePP communication model to minimize unnecessary data transfers: since event type **C** has a lower event rate, it is pushed directly to node 1. After reaching the pull-request (event type **C**), the event **B** is pulled to the source nodes [21]. Finally, the completed evaluation of the Query takes place in the central cloud. This selective combination of push and pull communication and operator placement significantly reduces the number of transmitted events while preserving semantic correctness.

The combination of optimized operator placement (INEv) and push-pull communication (PrePP) in the INES framework leads to substantial improvements over centralized CEP approaches. According to experimental results by Mr. Meran et al. [21], INES reduces the data transmission by up to 90%. Rather than forwarding all raw events to the cloud, the system distributes query processing across the network—bringing computation closer to the data sources and thus improving scalability and efficiency in fog-cloud environments [5, 21].

Despite these benefits, the INES Framework is limited to Single-Node-Placement. An operator is only placed once in the topology, neglecting the opportunity to place an operator multiple times in the topology [4, 5, 6]. Another crucial point is the time intensive computation time, which takes longer, the larger our network gets. His experiments showed, that a network size of 1000 nodes with maximal parents of 8, require a computation time of approximately three hours [21]. These limitations motivate the adoption and adaptation of MNP to our specific fog–cloud setting.

While multi-node placement (MNP) is a well-established concept in the literature, existing approaches typically operate in flat, (near) fully connected network topologies [5, 6]. In such environments, nodes can communicate directly, and MNP is often effectively realized by anchoring operators at source nodes. This design simplifies placement decisions and load balancing, but assumes direct connectivity and uniform node capabilities—conditions that do not hold in hierarchical fog-cloud environments.

In contrast, our work adapts MNP to a hierarchical fog-cloud environment, where event data flows upstream from the edge through one or more fog layers to the cloud. Here, **source nodes are not valid placement targets**, and placement must be performed on intermediate fog nodes or the cloud, respecting routing constraints, hop-based transmission costs, and heterogeneous node capacities. This upstream-oriented placement requires re-formulating the MNP logic to:

- consider only fog/cloud nodes on the shortest upstream paths that cover the required event bindings,
- integrate load- and resource-awareness by reducing available computational capacity when assigning operators,
- distribute partition able operators across multiple candidate nodes for load balancing,
- and maintain semantic correctness despite multiple simultaneous placements.

A further challenge is the computation time for finding optimal operator placement and push-pull edges. INES already exhibits long execution times for large networks, and the additional complexity introduced by MNP can further increase runtime. To address this, we incorporate a simplification-based graph compression technique that filters irrelevant nodes prior to placement, aiming to reduce the search space and improve scalability without compromising placement quality.

These problem definitions form the basis for the design of our INES* extensions, which are presented in Chapter 4.

4 Multi-Node Operator Placement for Fog-Cloud Environments

This chapter presents our approach to efficient operator placement in fog-cloud environments, focusing on *multi-node placement (MNP)* [6], its integration with existing single-node strategies to maintain efficiency and the reduction of the calculation time in order to find a nearly optimal execution plan. In section 4.1, we highlight the obstacles of the multi-node placement and introduce in section 4.2 several enhancements implemented in the INES* framework, including a simplified placement algorithm [26] and node characteristic constraints for operator placement. Our implementation of INES*, including all algorithms shown in this chapter, is publicly available on GitHub[1].

4.1 Obstacles of Multi-Node Operator Placement Approach

The design and implementation of a multi-node operator placement approach must account for several structural differences between our system and assumptions made in related works.

Network Structure

1. **Hierarchical Topology:** The approach assume a flat, undirected network where nodes can freely communicate with one another, while our system is based on a hierarchical fog-cloud environment. The Communication is directed and goes upstream from the edge layer to the fog, and finally to the central cloud node. This restricts the placement flexibility and introduces additional challenges for data flow and coordination.

2. **Node Heterogeneity:** Additionally, the approach showed a homogeneity of the nodes in the network. The nodes in our simulation environment consists of distinct nodes with different computational power and memory depending on hierarchy level of the nodes.

[1]`https://github.com/ND-2002/INES`

These obstacles had to be taken into account during implementation to ensure the correctness of our approach.

4.2 Improvements in INES*

In this section, we present the technical improvements implemented in the INES* framework. The enhancement in section 4.2.3 enables the execution of existing methods in a more efficient and realistic manner by taking the computational power of the nodes into account. This improvement is addressing **RQ1** and **RQ2**. In order to answer **RQ3**, we introduce an approach in which we reduce the computation time for finding execution plan through graph compression techniques in section 4.2.2.

4.2.1 Multi-Node Integration

To support distributed query execution in INES*, we extend the operator placement module to integrate the **Multi-Node Placement** (MNP) strategy into the existing single-node (SNP) framework. The goal is to allow a subquery to be evaluated across multiple fog nodes, enabling early partial computation and thus reducing upstream transmission costs. Our integration provides a mechanism for decomposing a Query into multiple subqueries based on shared event types [6].

Algorithm 1: Interaction between MNP and SNP in `calculate_operatorPlacement()`

```
Parameters: partType: anchor event type (if partitionable)
partType ← returnPartitioning(...)    // anchor event type if projection is partitionable
// If partType exists, use MNP; otherwise SNP
if partType ≠ ∅ then
    result ← computeMSplacementCosts(...)        // Multi-Node Placement (MNP)
end
else
    result ← ComputeSingleSinkPlacement(...)     // Single-Node Placement (SNP)
end
```

Placement Decision Logic. For each operator, we determine its suitability for multi-node placement. As in our algorithm 1 shown, this is done using the function `returnPartitioning`, which attempts to identify a suitable `partType`, which is a dominating event type that allows to partition the operator across multiple nodes. This logic reflects the concept of anchor-based decomposition introduced in the MuSe paper [6].

If a `partType` is found, the projection is distributed across the nodes where this event type is locally available. This triggers our customized method `computeMSplacementCosts`, which computes the optimal set of partial placements. Each partial operator instance processes its local share of the data, and only the intermediate results are forwarded, thereby minimizing the overall transmission costs.

If the partitioning is not possible, i.e. `partType` is empty, the system falls back to `ComputeSingleSinkPlacement`, placing the operator on a single node that can access all required input streams.

Implementation Challenges. The Integration of both strategies into the operator placement function, required several architectural decisions. The challenge was to maintain a correct separation between MNP and SNP strategy, especially as both placements require cost computation and resource checks. To overcome these difficulties, we implemented a decision layer within the calculate_operatorPlacement function (defined in our operatorplacement.py class).

This logic delegates to the appropriate placement strategy, which selects either SNP or MNP based on the result of the `returnPartitioning` function.

Another important task was to manage the intermediate states and ensuring correctness when projections are split across multiple nodes. This involved adapting the internal data structures for mapping operators to the nodes and ensuring that event bindings are preserved across partial placements.

Key Variables/Functions:

1. **Variable** `partType`: The anchor event type is used to determine, if a projection can be partitioned across multiple nodes. This is derived from the event binding characteristics of the operator.

2. **Function** `computeMSplacementCosts`: A function that evaluates the cost of distributing an operator across multiple nodes.

3. **Function** `ComputeSingleSinkPlacement`: If no partitioning is feasible, we assign an operator to a single node.

This integration enables INES* to flexibly assign operators depending on query semantics and data locality.

4.2.2 Simplification-Based Method in INES*

To address **RQ3** What are the drivers of exhaustive computation time in INES, and how can it be improved?, INES* implements graph compression via relevance-marked nodes based on the **simplification-based method(2.4.3)**.

Finding an approximate optimal execution plan in fog-cloud networks can be computationally expensive due to the many nodes involved. For single-node placement, many fog nodes, however, merely forward events and do not participate actively in query processing. While with multi-node placement, multiple nodes process queries in the fog layer, which can cause a high resource consumption.

In order to reduce the complexity of the operator placement algorithm, we introduce a graph compression technique that is configured as follows:

- A dictionary in which each node is assigned the set of event types that pass through it (nodes as keys, event types as values).
- Nodes with more than one event type traversing through them are considered relevant for operator placement, as they could serve as potential placement candidates. Fog nodes that only forward a single event type are marked as non-relevant and excluded from the operator placement search space.
- Despite the compression, the complete topology is retained for path and cost calculations to ensure accurate data transmission and hop counts modeling.

Algorithm 2: Graph Compression

```
Parameters: $G$: The graph of the network
            eList: Dictionary with nodes as keys and their events as
values
compList ← empty list
// Add relevant nodes into the compressed graph list
for (node, etypes) in eList.items() do
    if len(etypes) $\geq 2$ then
        append node to compList
    end
end
compressed_nodes ← sorted set of compList
// Mark relevant nodes for operator placement
for n in G.nodes do
    if n in compressed_nodes then
        mark n as relevant
    else
        do not mark n as relevant
    end
end
```

4.2.3 Node Characteristic Constraint for Operator Placement

To answer **RQ1:** How can multi-sink placement be efficiently integrated into INES? and **RQ2:** What is the impact of multi-sink placement on transmission cost compared to single-sink placement?, we introduce a **node characteristic constraint** that takes the computational power of each node into account. The nodes in our simulation environment are heterogeneous and contain the following characteristics:

- **ID:** Which is automatically assigned to the nodes at its creation. The numeration is consecutive.
- **Computational Power:** is the virtual capacity required to process projections and increases with higher hierarchical levels in the network topology.
- **Memory:** The virtual capacity to store processed projections. The Memory also increases with higher levels in the hierarchy.
- **Event Rates:** Rates of the primitive events which are stored in the leaf node. Represented as Array with integers. If no event type at node event rate marked as zero.

In addition to these characteristics, the node class features a flexible parent-child relationship model, allowing each node to have multiple parent and child connections [21].

New Constraint

In previous versions, sink nodes were selected solely on the basis of their available computing power, but once selected, their computing capacity remained unchanged. In our approach, the computational power of a node is now treated as a consumable resource. This means that every time a node is chosen as a sink, its computational power is reduced by the computational requirements of the assigned projection. This prevents overloading nodes and leads to a more realistic simulation of resource constraints in fog-cloud environments.

4.3 INES Key Parameters

The following section 4.3 introduces the primary configuration parameters within the INES framework. These parameters determine the network structure, event generation, and query complexity, and are essential for understanding how the system's topology and workload characteristics influence its performance and evaluation outcomes.

- **Network Size:** The value of the network size determines the number of nodes in the network. The more nodes, the more complex the topology becomes. The hierarchical structure is created using the base 2 logarithm of the network size.

- **Node-Event-Ratio:** This parameter specifies the distribution of each primitive event type to the source nodes. For example, if we have 5 event types and an node-event-ratio of 0.5, which means a probability of 50 per cent, each source node will receive the respective primitive event type with a 50 per cent probability. Only event-generating nodes are affected, which means that the fog and cloud nodes are not affected.

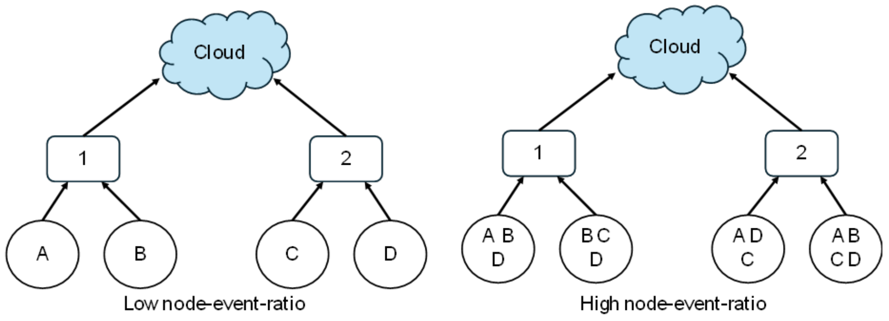


Figure 10: Graph with high and low Event-Node-Ratio.

The following figure 10 illustrates the difference between low and high node-event ratios. At low node event ratio, each source node has a small chance of generating any given primitive event type, resulting in fewer events overall and a sparser workload. With a high node event ratio, each source node is very likely to generate each event type, leading to more frequent events and a heavier workload per node. By comparing the low and high node-event ratio in Figure 10, we visually highlight how this parameter controls event density and distribution across the network. This becomes especially meaningful when evaluating system behavior (e.g., operator placement, latency, resource utilization) under varying workload conditions.

- **Number of Event Types:** This parameter defines the number of primitive events in a query. If this value is set 3, the event types are defined as 'A', 'B' and 'C'.

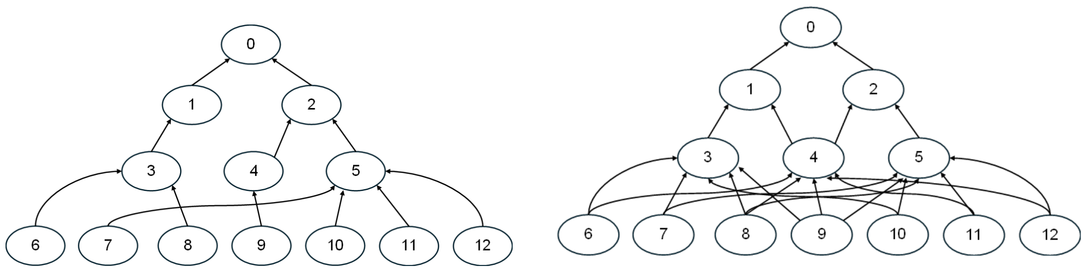
- **Event Skew:** The Zipfian parameter controls the skewness in the distribution of event rates for primitive events. If we have a lower event skew, the rates of the generating events are almost the same. The higher the value, the higher the disparities between the rates of the events.

Event rates		
Event type	**Event skew**	
	0.1	**2**
A	2000	21
B	2000	1330
C	2000	2000
D	2000	652

Table 2: Impact of event skew on event rates.

Table 2 is presenting the event rates generated per event type, depending on the Zipfian parameter. A lower event skew generating nearly similar event rates, while on the other hand a higher skew results causes significant disparities of generated event rates between each event type.

- **Maximal Parent:** This parameter defines an upper bound on the number of parent nodes that any given node may possess within the network. While a node can have up to this number of parents, the actual count may vary to preserve the stochastic nature of the network topology. Higher values for this parameter generally lead to a denser graph structure by increasing node interconnectivity.



(a) Graph with low density. (b) Graph with high density.

Figure 11: Figures with different graph density.

Figure 11 illustrates a network of 13 nodes whose density varies with the *max_parents* setting. With *max_parents*=1, the topology is relatively sparse (Fig. 11a), whereas increasing it to *max_parents*=3 results in a significantly denser network, with a higher number of connections between nodes (Fig. 11b).

- **Query Size:** Specifies the total number of queries to be generated and evaluated during the simulation.
- **Query Length:** Defines the maximum structural complexity of a query, typically measured by the number of primitive events it contains.

Query Length		
Query	**Low Complexity**	**High Complexity**
Query 1	$\textbf{AND}(A, F, C)$	$\textbf{AND}(N, P, B, \textbf{SEQ}(H, I, M))$
Query 2	$\textbf{SEQ}(G, E, F)$	$\textbf{SEQ}(E, O, \textbf{AND}(F, \textbf{SEQ}(U, T, S)))$

Table 3: Example of different Complexity by increasing Query length.

Table 3 visualizes different complex queries we can realize by changing the **query length** parameter.

4.4 Query Splitting with Multi-Node Placement and Push-Pull Communication

In our thesis we enhance INES with the following features, enabling multiple placements in the topology 4.2.1, a new and slightly more realistic design of our simulation environment, by reserving nodes in the fog layer for a placement, if the computation power is given 4.2.3. In addition, we reduce the topology by implementing a graph compression technique 4.2.2.

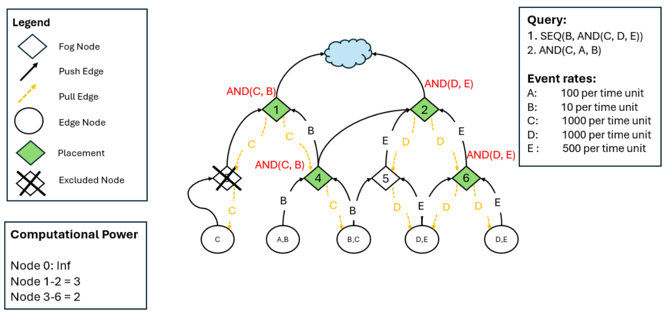


Figure 12: INES* Features.

Figure 12 visualizes the key enhancements introduced in this work, using a complex example with two distinct queries: `SEQ(B, AND(C, D, E))` and `AND(C,` `A, B)`. The event types involved have heterogeneous rates per time unit: **C** and **D** occur most frequently with 1000 events per unit, followed by **E** with 500, then with **A** (100), and finally **B** (10). The network topology comprises five edge nodes in an edge layer, a two-layer fog structure consisting of nodes 3–6 in the first fog layer with computing power 2 and nodes 1–2 in the second layer with power 3, and finally a central cloud node. The operators `AND(C, B)` are strategically placed at nodes 4 and 1, and `AND(D, E)` at nodes 2 and 6.

The placements respect the available computational power of each node according to our constraint-aware placement strategy 4.2.3 in order to prevent node overloading. By distributing operators across multiple fog-layer nodes, both queries achieve lower transmission costs compared to single-node placement, as computations can be performed closer to the data sources. For example, Node 4 can host at most one operator due to its computational limit. Although both `AND(C, B)` and `AND(A, B)` are eligible for placement, only `AND(C, B)` is deployed.

After operator placement, INES* applies the predicate-based push-pull communication model. The event type with the lowest rate triggers a pull request, by pushing this event to the highest placed operator for the corresponding query. All upstream node then push higher-rate event types back to the source nodes. This strategy minimizes redundant transmission by pulling only what's needed. For example, edge nodes push event type **B** to Node 1, in response, after receiving the pull request, event type **C** will be pulled to the source nodes.

To reduce the search space and avoid placing operators in irrelevant areas, our simplified-based graph compression technique(see 4.2.2) is applied. Node 3 is excluded from placement, as only the one single event type **A** passes through it.

4.5 Research questions to be examined

In this work, we investigate the impact of our extended INES* framework with our introduced features. Based on our implementation, we formulate the following assumptions to be evaluated in the experiments presented in Chapter 5.:

- We reduce the INEv and overall transmission costs.
- The push-pull latency will increase due to the number of placements in the topology.
- Our graph compression technique reduces the required overall computation time.
- The node constraint through computational power, increases transmission costs, leading potentially to an overhead, as the multi-node placement logic cannot be applied correctly.

5 Evaluation

In this Chapter we conduct several experiments in order to examine our implemented solutions and evaluate the results. In Section 5.1 we introduce our experimental setup, describing the *data and workload* and *evaluation metrics*, and in Section 5.2 we conduct our experiments.

5.1 Experimental Setup

Data and Workload

We use fictitious data and workloads for our experiments that are not derived from real-world data. The events are shown as letters A, B, C etc.. The workload is composed of different queries made up of combinations of these events, where the size of the queries and the complexity are systematically varied to evaluate the performance of the system in different configurations.

Evaluation Metrics

- **Transmission Ratio:** This metric is calculated relative to a centralized cloud-based baseline. It quantifies the total volume of data transmitted within the network in comparison to the baseline, thereby providing insight into the effectiveness of the proposed approach in minimizing communication overhead across the fog-cloud hierarchy.

- **Network Latency:** In the simulation environment, latency is measured in terms of hop count. Each traversal across an edge in the network graph increments the total latency by one unit, offering a simplified yet effective model of measurement within the system.

- **Computation Time:** Denotes the time required by the algorithm to compute an optimal solution for a given network configuration. It reflects the algorithm's efficiency and scalability with respect to problem complexity.

- **INES SNP:** Refers to the system before the new implementation. This includes operator placement with single sink placement and push-pull communication (SNP Hybrid).

- **INES SNP_{GC}:** Refers to the system before the new implementation. This includes operator placement with single sink placement and push-pull communication enhanced with the graph compression technique.
- **INES* MNP:** INES* refers to the system with the extension of the system with multi-node placement in operator placement and push-pull communication (MNP Hybrid).
- **INES* MNP_{GC}:** Refers to the extended system (INES*) enhanced with a graph compression technique. This optimization aims to reduce computational complexity and improve execution time to find a sufficient execution plan.

Computing Environment

We compute in a simulation environment with fictitious data. The Simulation allows us to simulate different data characteristics (4.3).

Hardware and Software Settings

Our Experiment is on a university-hosted cluster, which has 60 GB of RAM and runs a Linux-based operating system. It is accessible via SSH.

5.2 Experiments: Design and Interpretation of the Results

In this section we present our experiments in order to answer our research questions and the impact of our approach on the simulation environment. First, we investigate the computation time of INES in Section 5.2.1 and evaluate the effect of our graph compression on INES and our INES* approach (see Section 5.2.2). Next, we evaluate our approach on varying graph density (Section 5.2.3), influenced by the variation of our key parameters(visible in Section 4.3) and contrasts the results with the resource-constraint defined in Section 4.2.3. Finally, we compare INES* to INES in terms of network transmission cost and hop-based delay under different graph configurations 5.2.4.

5.2.1 INES Computation Time

In this subsection we benchmark and compare the execution time of the exhaustive INES Baseline Framework under different configurations. In particular, we analyze how the runtime scales with the number of network nodes and the complexity of the query, which is defined by the number of queries, their length, and the number of event types.

Impact of network size

For the network size evaluation, we simulate environments with 10, 50, 100, 150, 200, 250, and 300 nodes (Figure 13). All other parameters, including the number of events and query structure remain fixed during these runs.

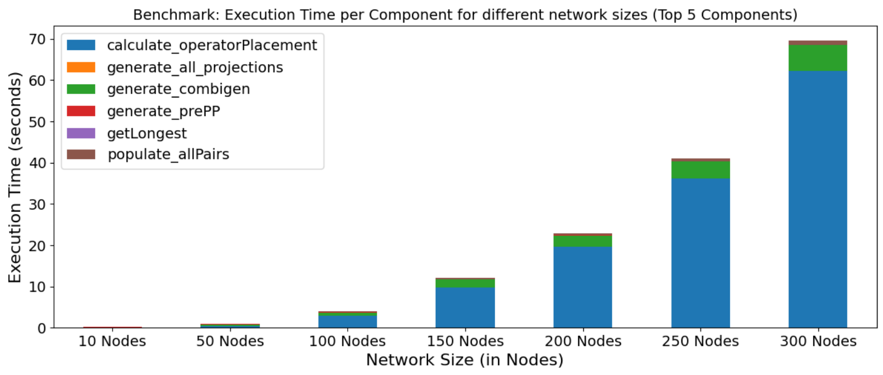


Figure 13: INES SNP: Execution Time for different Network Sizes.

Observations In the Figure 13 wee see First, that the total execution time increases significantly with the number of nodes in the network. Second, whereas time for *generate_all_projections*, *generate_prePP* and *generate_combigen* is constant or minor affected, operator placement rises sharply. Already at 100 nodes, it becomes clear that the operator placement takes up a lot of time and increases to over 62 seconds, accounts for approximately 89% of the total calculation time at 300 nodes.

Result The computation is heavily affected by the node size. This is due to the growing number of candidate nodes in the fog layer that must be considered for each operator placement. This becomes particularly clear in the function *calculate_operatorPlacement*, in which the actual placement decisions are made. To mitigate these computational challenges and enhance scalability for larger networks, we introduce a graph compression strategy (4.2.2). By reducing the effective size of the network topology, we aim to lower the number of possible placement candidates and thereby reduce the runtime of primarily *calculate_operatorPlacement*.

Impact of query complexity

For analyzing query complexity, we begin with a base configuration consisting of 6 event types, 3 queries, and a query length of 4. This setup is denoted as `6E_3Q_4QSIZE` in Figure 14. In subsequent tests, we increment each of these parameters by one in four successive steps, resulting in progressively more complex workloads while keeping the network size and other key parameters constant.

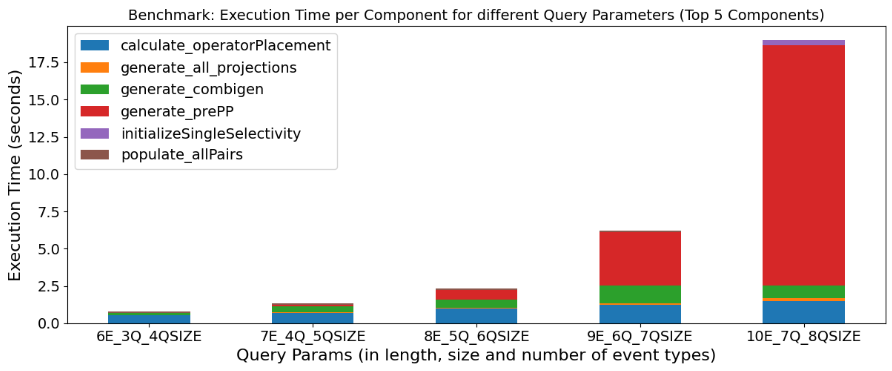


Figure 14: INES SNP: Execution Time for different Query Parameters.

Observations In scenarios where the complexity of the input query is increased, by extending the number of projections, their length, and the number of event types (Figure 14), the *generate_prePP* function becomes our second discovered computational bottleneck. This time the computation time of *calculate_operator-placement* increases slightly together with *generate_combigen* and *generate_prepp* up to the workload `8E_5Q_6QSIZE`, while the other functions are constant or minor affected. Starting from `9E_6Q_7QSIZE`, the *generate_prePP* function begins to stand out here. In the most complex configuration `10E_7Q_8QSIZE`, it becomes the most time-consuming component, contributing over 87% of the total execution time with an average runtime of more than 16 seconds.

Result While *calculate_operatorPlacement* scales with network size, the *generate_prePP* function is affected by the complexity of the query. This is due to the exhaustive generation of all possible push-pull plans. This involves evaluating all valid event projections across all nodes and computing exact cost estimates for each configuration, which scales with the number and size of event types involved in the queries.

Conclusion

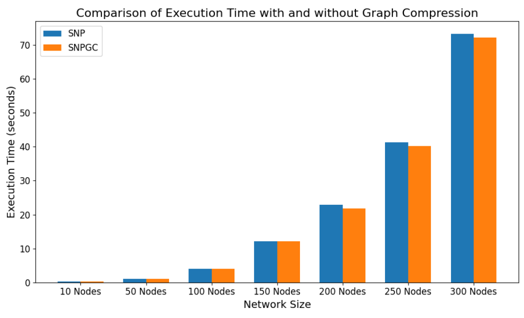
Although our graph compression approach (see Section 4.2.2) aims to reduces the number of placement candidates and thereby optimizes *calculate_operator-Placement*, it does not directly affect the *generate_prePP*. As the core of this thesis operator placement, by extending it with multi-node placement, our primary optimization effort focuses on mitigating the computational overhead caused by increasing network size. Although *generate_prePP* contributes to the total computation time under high query complexity, it remains out of the scope of this work. One reason is that OP and PP are currently computed sequentially, which might introduces additional overhead. A joint, interleaved computation of both steps could yield further performance improvements; however, this approach is part of ongoing research and is being pursued in a parallel thesis project by Mr. Glück.

5.2.2 Effect of Graph Compression on Computation Time

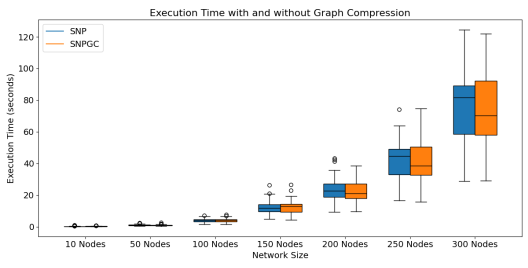
In this subsection, we analyze the impact of the proposed **graph compression technique** on the overall computation time of the INES framework. Specifically, we compare the runtime of the baseline INES (SNP) and INES* (MNP) configurations with and without graph compression, across increasing network sizes.

Impact of Graph Compression on INES

In this experiment we investigate the impact of our graph compression on the network size of the INES-Baseline Framework. The different network sizes were run 100 times each for our tests. The results presented in Figure 15 demonstrate the comparative performance of the INES framework with (SNP_{GC}) and without graph compression (SNP). We use the same parameters as described in subsection 5.2.1.



(a) Comparison SNP and SNP_{GC}.



(b) Comparison of Data Scattering.

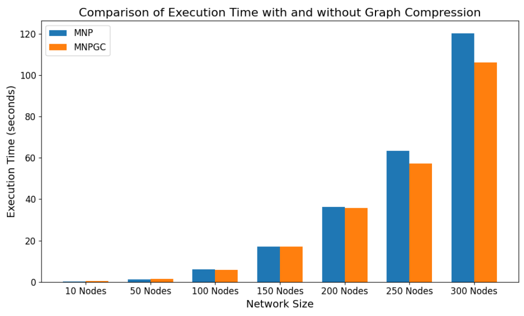
Figure 15: Comparison of Computation Time INES SNP and SNP_{GC}.

Observations Subfigure 15a shows a slight but consistent reduction of approximately 3% in average computation time for SNP_{GC} starting from a network size of 200 nodes. For smaller network sizes (10 to 150 nodes), the computation times for SNP and SNP_{GC} remain nearly identical. Subfigure 15b further illustrates that the median improves at 150 nodes onward and is up to 12.5 lower at 300 nodes. Another observation is that the variance in execution time increases substantially with the network size. At 300 nodes, for instance, we have outliers of up to about 30 seconds.

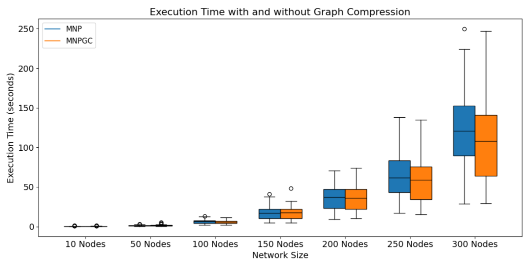
Result We achieved with the implementation of the simplified-based graph compression technique a slightly beneficial effect of around 3% on the total computation time. One reason for this is that the single-node placement strategy performs fewer placement iterations, as only one possible placement is considered per query. Due to the graph compression, nodes, especially in the fog layer, are filtered out early, as only a single event type traverse the node, which means that the placement condition cannot be fulfilled. This reduces the number of candidate nodes and, consequently, the overall number of iterations.

Impact of Graph Compression on INES*

Now, we analyze the impact of our graph compression technique on the network size of our INES* Framework. In Figure 16, we see the comparison of our INES* Framework with the implemented technique (MNP_{GC}) and without the graph compression (MNP).



(a) Comparison INES MNP and MNP_{GC}.



(b) Comparison of Data Scattering.

Figure 16: Comparison of Computation Time INES MNP and MNP_{GC}.

Observations We first see in subfigure 16a that for smaller network sizes (10 to 150 nodes), the total computation time remains nearly identical for both MNP and MNP_{GC}. However, starting at 200 nodes, the benefit of graph compression becomes more apparent, with MNP_{GC} consistently outperforming MNP. The most notable improvement is observed at 300 nodes, where MNP_{GC} reduces the total execution time by approximately 14%. Subfigure 16b highlights the increased variance in computation time as the network size grows. This variance is especially pronounced at 250 and 300 nodes, where outliers indicate occasional long runtimes.

Result The observed performance gain is mainly due to the reduced number of candidate nodes considered during the placement process. As the network grows, graph compression becomes increasingly effective by eliminating nodes that are not suitable for operator placements. This leads to fewer iterations per projection and ultimately reduces the overall computation time. Consequently, the benefit of graph compression becomes more significant with larger network sizes as our experiment shows.

Conclusion

Overall, the results of our experiments demonstrate that our proposed graph compression technique consistently reduces the computation time of the INES framework, particularly in larger network topologies. While the performance remains nearly the same for the network sizes between 10 and 150 nodes, its effect becomes slightly visible for the single-node operator placement strategy (SNP) and increasingly visible for the multi-node placement (MNP) as the network size grows beyond 200 nodes. In both cases, graph compression contributes to more efficient execution time by reducing the number of eligible nodes early in the process and eliminating irrelevant candidates during operator placement. Our findings confirm that graph compression is an effective preprocessing step to enhance the runtime performance in distributed event processing systems, especially in large-scale networks. However, further analysis and refinements to the compression logic may be required to fully exploit its potential in large-scale, heterogeneous networks.

5.2.3 INES* Varying Density Configurations of the Graph

In this experiment, we evaluate different density configurations of the network graph by varying the *maximum number of parents* per node. We consider query workloads of size 5 and 10 and analyze the impact along several key parameters. Subsequently, we compare the performance of our INES* framework both with and without resource constraints 4.2.3. Furthermore, we evaluate the behavior of the MNP in comparison to the SNP across varying conditions. Finally, we examine the effect on the latency across different *network sizes* and *max parents*.

INES* Impact on the Transmission Costs

We begin by analyzing the performance of INES* with operator placement (MNP) and after executing push-pull communication ($MNPHybrid$) under increasing graph density by varying the key parameters. These include a fixed network size of 50 nodes, different numbers of event types (ranging from 5 to 15), and varying query workloads (5 and 10) for different configurations of maximum parents (see Figure 17). Additionally, we examine workloads of sizes 5, 10, 20, and 25 under a fixed number of 25 event types in Figure 18. Further parameters include event skew values ranging from 0.1 to 2.0 as shown in Figure 19, and node-event-ratios between 0.1 and 1.0 (see Figure 20). The following figures illustrate the behavior of the *transmission ratio* across these varying conditions.

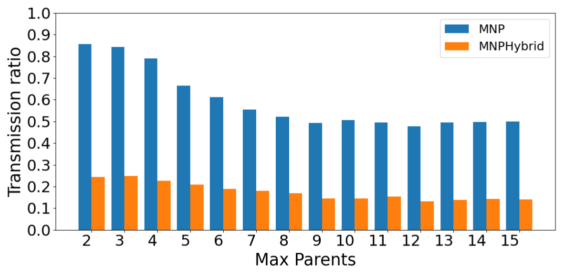
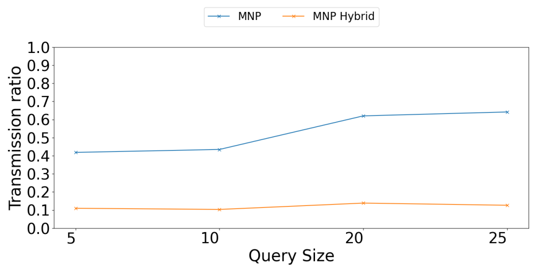
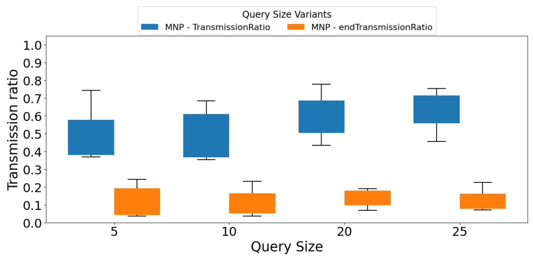


Figure 17: MNP Transmission Ratio along different Max Parents configurations.

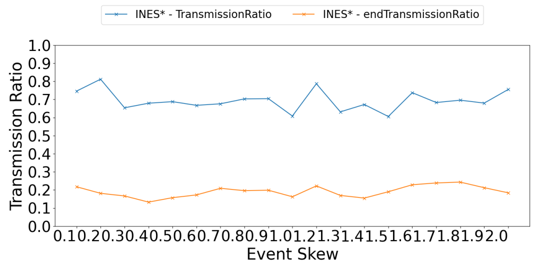


(a) Query Size Transmission Ratio.

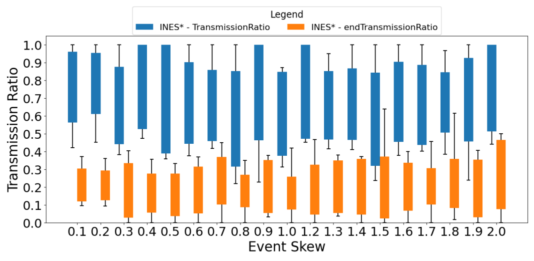


(b) Query Size Scatter.

Figure 18: Comparison MNP Transmission Ratio along different Query Sizes.

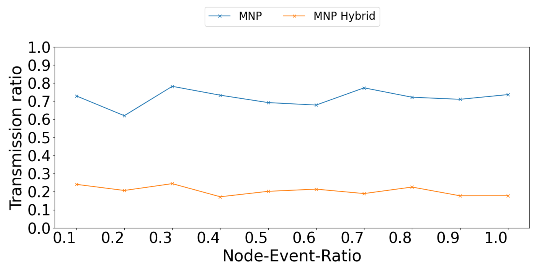


(a) Event Skew Transmission Ratio.

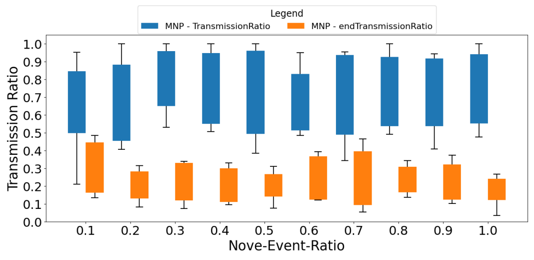


(b) Event Skew Scatter.

Figure 19: Comparison of MNP Transmission Ratio along different Event Skews.



(a) Node-Event-Ratio Transmission Ratio.



(b) Node-Event-Ratio Scatter.

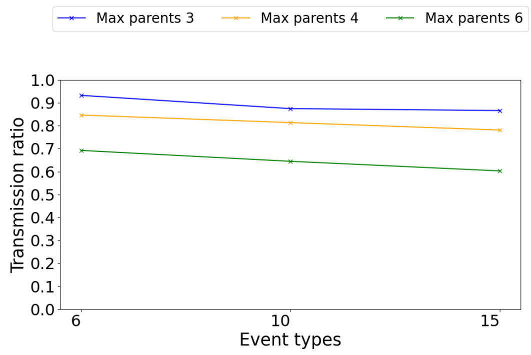
Figure 20: Comparison of MNP Transmission Ratio along Node-Event-Ratio.

Observations Our Evaluation shows in Figure 17 that the operator placement strategy with MNP reduces the transmission costs compared to the central costs by increasing the *Max Parents* parameter. This is due to the fact that there are more routing options due to the density and therefore more placement options further down in the topology, which saves costs. After applying PrePP, the transmission cost decreases significantly. In Figure 18a, we observe the impact of increasing query size on transmission costs, with a fixed *maximum number of parents* of 4 and a fixed *number of event types* of 25. For the MNP strategy, the transmission costs increases steadily from around 0.42 at a query workload of 5 to approximately 0.65 at size 25. In contrast, the MNP *Hybrid* variant maintains a significantly lower and nearly stable transmission ratio across all workload sizes, highlighting the effectiveness of the hybrid logic in distributing operators more efficiently across multiple nodes. Figure 18b provides its box plot analysis. Here, the distribution confirms the trend. MNP results in a wider spread and overall higher transmission costs, whereas MNP *Hybrid* consistently achieves lower medians and narrower spreads. This indicates that the hybrid multi-node strategy not only reduces average transmission overhead but also leads to more predictable performance across varying query complexities. In Figure 19, we analyze event skew between 0.1 and 2.0. We observe a slight degradation in transmission efficiency as skew increases, particularly for MNP, while the hybrid variant demonstrates greater robustness. Finally, Figure 20a (supported by the boxplot in Figure 20b) reveals that increasing the node-event ratio (from 0.1 to 1.0) improves transmission costs. This is expected, as a higher node-event ratio implies more possible placements per event, allowing for more optimized operator placements.

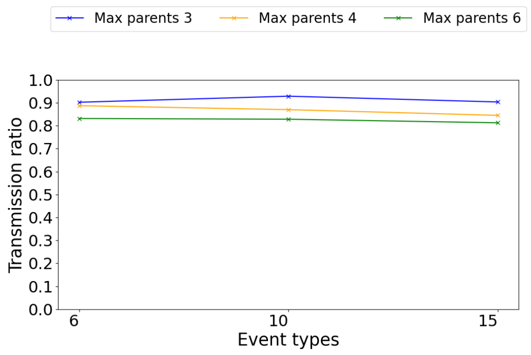
Result Our evaluation shows that the observed improvements in transmission costs are primarily attributed to the use of our multi-node placement strategy (MNP), which significantly reduces data movement by distributing operators more flexibly across the network, compared to the centralized costs. This effect is further amplified when applying our PrePP optimization, as seen in multiple figures. Among the tested parameters, the *Max Parents* configuration had the most substantial impact. As graph density increases, the transmission ratio drops considerably due to a wider range of placement options, allowing operators to be placed closer to data sources. The improvement becomes particularly noticeable beyond a Max Parents setting of 3 or 4 and remains approximately the same at 50 per cent from 8 nodes upwards. In contrast, varying the query workload size (Figure 18a and 18b) revealed a consistent increase in transmission ratio with larger workloads, yet the relative advantage of MNP and $MNPHybrid$ remains stable—demonstrating good scalability. The statistical distribution in the boxplots confirms that $MNPHybrid$ handles larger workloads more gracefully. Parameters like event skew and node-event-ratio showed moderate to limited impact. While high skew levels (Figure 19) can reduce the effectiveness of placement strategies due to localized data flow, our $MNPHybrid$ approach still maintains stable performance. Node-event-ratio variations (Figure 20) affected the system less significantly, though higher ratios slightly improved placement flexibility. In summary, our multi-node placement strategy combined with PrePP consistently outperforms the centralized baseline across various topological and workload conditions.

Impact of Resource-Constrain on INES*

In this experiment we compare our extension INES* with the *node characteristic constraint* for our operator placement strategy (see Section 4.2.3). For the comparison, we use the same configurations mentioned in Section 5.2.3.

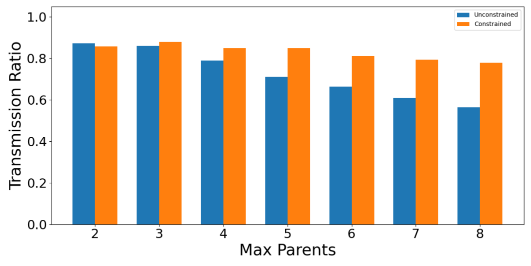


(a) INES* without resource-constrain.

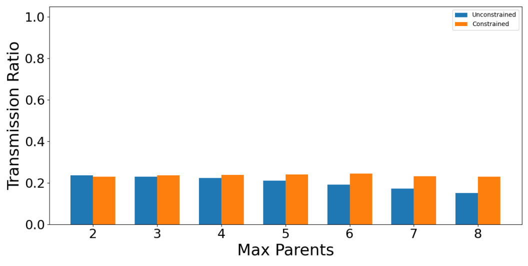


(b) INES* with resource-constrain.

Figure 21: Comparison Resource-Constrain on INES* along different event types.



(a) INES* before Push-Pull.



(b) INES* after Push-Pull.

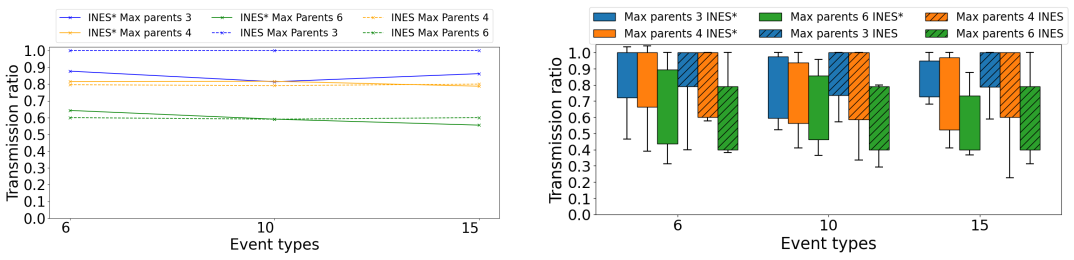
Figure 22: Comparison Transmission Ratio along different Max Parents configurations.

Observations Figure 21 is a line graph and illustrates the impact of our resource constraints on the transmission ratio for INES*, evaluated over different numbers of *event types (6, 10, 15)*. Subfigure 21a shows the performance without the constrain, where we observe that increasing the number of event types results in a slight but consistent decrease in transmission ratio across all configurations of maximum parents. Moreover, increasing the Max Parents parameter improves the transmission ratio further. In contrast, Subfigure 21b reveals that, under resource-constrained conditions, the overall transmission ratio remains consistently above 0.8 on average, showing only marginal improvements as the number of event types increases. In Figure 22, we compare the transmission ratio across different maximum parent configurations before and after applying *push-pull communication*. Again, before (22a) as well as after applying *push-pull communication*(22b) the unconstrained variant achieves significantly lower transmission ratios, particularly as the number of allowed parents increases. The constrained setup maintains consistently higher values, indicating that resource restrictions hinder the system's ability to optimize communication efficiency as graph density grows.

Result These results highlight that unconstrained operator placement enables significantly more efficient data transmission, especially in denser graphs and more complex workloads. By not being limited by resource availability, INES* can strategically place operators to minimize inter-node communication, using the multi-node strategy logic without any restriction. In contrast, the constrained setup is forced to place operators on available but potentially suboptimal nodes, resulting in increased data transmissions. While resource awareness is necessary in practice, it introduces a trade-off in efficiency that becomes more evident as the complexity of the network and event workload increases. The resource constraint approach brings our simulation closer to real-time scenarios, but has to be further adapted with regard to the multi-sink logic in which the same placements are shared.

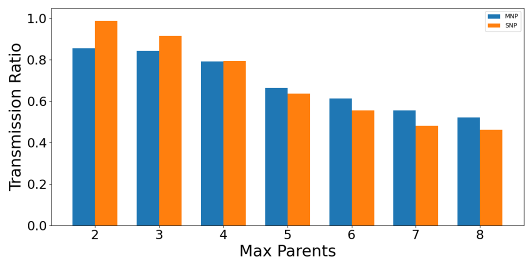
5.2.4 INES* Comparison with INES

In this experiment, we compare our multi-node placement approach (MNP) to the single-node placement strategy (SNP), analyzing how both behave under varying key parameters.

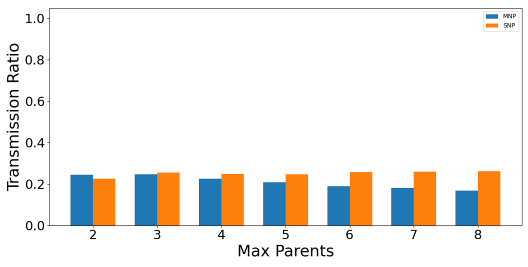


(a) (b)

Figure 23: Comparison of Transmission Ratios along different Max Parents configurations.



(a) Base Transmission Ratio.



(b) Transmission Ratio Hybrid.

Figure 24: Comparison of MNP and SNP Transmission Ratio along different Max Parents configurations.

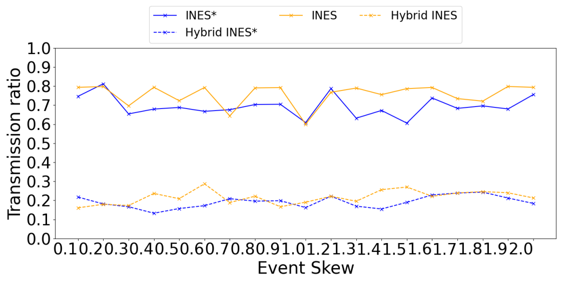
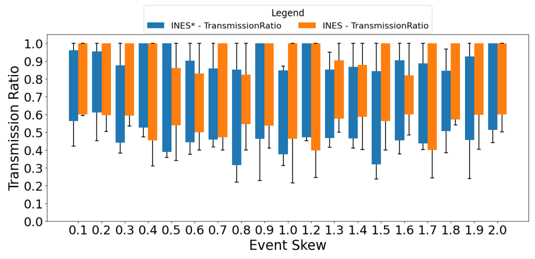
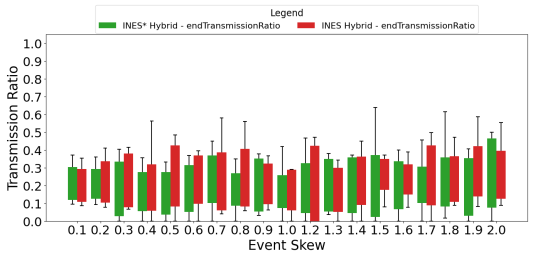


Figure 25: Transmission Ratio along different Event Skews.



(a) Base Transmission Ratio.



(b) Hybrid Transmission Ratio.

Figure 26: Comparison of MNP and SNP Transmission Ratio along different Event Skew configurations.

Observations Figure 23 visualizes the impact of increasing graph density, represented by the *Max Parents* parameter and the *number of event types*, on the transmission ratio. Both subfigures indicate that INES* (MNP) outperforms INES (SNP) for denser graphs starting at 10 event types. The improvement becomes more visible in Figure 23b, where the boxplots demonstrate a clearer reduction in median transmission ratio for INES* across all configurations. This reduction is more pronounced at Max Parents value of 3.

Figure 24 further shows a different sight. Before applying push-pull communication *MNP* outperforms *SNP* till a max parents size of 3, whereas with a max parent size of 5, *SNP* has lower transmission costs(Figure 24a). After applying our hybrid optimization through PrePP (Figure 24b), the gap between the two strategies widens considerably, showing the advantage of combining *MNP* with PrePP logic.

When analyzing the influence of event skew, Figure 25 shows that INES* generally achieves better transmission efficiency than INES across the entire skew spectrum. This trend is supported by the boxplots in Figure 26, where the average transmission ratios for INES* remain consistently lower than for INES in both base and hybrid scenarios. Moreover, INES shows more variability and higher worst-case values, indicating less robustness to skewed event distributions.

Result The comparison between INES* and INES clearly demonstrates the advantages of our multi-node placement strategy over the single-node placement strategy. Especially under increasing graph density, INES* achieves significantly lower transmission ratios, confirming that our approach benefits from the additional flexibility in operator placement.

These improvements are further amplified after applying the PrePP optimization, which consistently outperforms *SNP*. In particular, the hybrid variant of INES* shows a substantial reduction in transmission ratio across all Max Parent configurations (Figure 24b), achieving up to 37% lower transmission ratio compared to *SNP* at *Max Parents = 8* (0.17 vs. 0.27). In the base setup without PrePP, INES* still performs better in lower-density configurations (e.g., Max Parents = 2–4), but *SNP* begins to slightly outperform *MNP* at higher densities (Figure 24a).

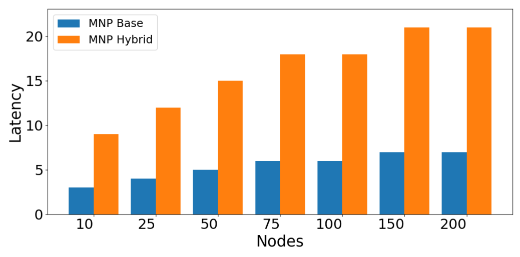
When comparing transmission ratios across different numbers of event types, INES* also consistently outperforms INES at a higher complexity starting at a event type number of 10. As shown in Figures 23a and 23b, the transmission ratio for INES* decreases with increasing event types, particularly at higher Max Parent configurations. The boxplots further confirm that INES* not only reduces average transmission cost but also demonstrates lower variance, indicating more consistent communication performance.

While both strategies are influenced by event skew, INES* maintains lower transmission costs and demonstrates more stable behavior under skewed conditions (Figures 25 and 26). This robustness illustrates that *MNP*, especially when combined with PrePP, is not only more efficient but also more adaptable to varying workload distributions.

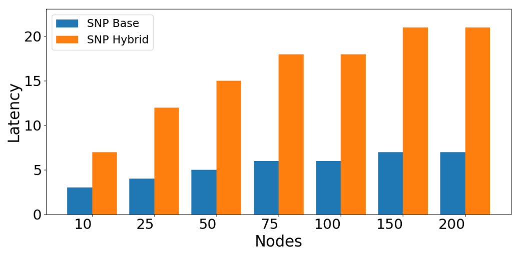
Overall, these results validate our enhancements to the INES framework and highlight the significant efficiency gains enabled by MNP. Nevertheless, the results also indicate that in mid to highly connected topologies (e.g., Max Parents > 4), the single-node strategy may still yield slightly better results, pointing to optimization potential for our approach in hierarchical fog-cloud environments.

INES* Impact on Latency

In this part of the evaluation, we focus on comparing the latency under varying network configurations. Although MNP reduces the transmission costs for operator placement, we assume that it increases latency. To examine our hypothesis, we analyze how our multi-node placement strategy (MNP) and the single-node placement strategy (SNP) perform in terms of latency, both in their base versions and after applying PrePP. The latency is measured along two parameters: increasing network sizes (with a fixed maximum of one parent per node) and increasing graph density (via the Max Parents parameter).

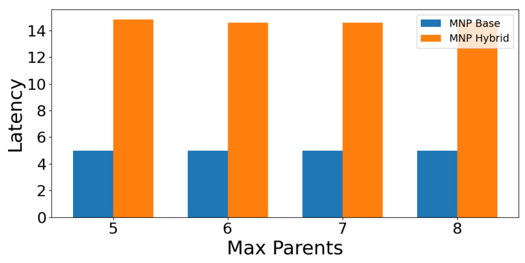


(a) MNP Network Latency.

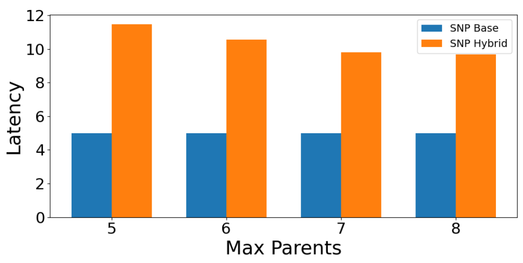


(b) SNP Network Latency.

Figure 27: Comparison MNP and SNP Latency along increasing Network Sizes with Max Parent = 1.



(a) MNP Max Parents Latency.



(b) SNP Max Parents Latency.

Figure 28: Comparison MNP and SNP Latency along Max Parents.

Observations Figure 27 shows the development of latency for MNP and SNP under increasing network sizes, with a fixed Max Parents value of 1. In both strategies, latency increases as the number of nodes grows. However, the hybrid variants (with push-pull communication) lead to a much steeper increase. For instance, at 200 nodes, the latency for $MNPHybrid$ reaches over 21 units, while the base remains below 8. The same trend holds for SNP, though at slightly lower absolute values.

Figure 28 presents latency behavior across different Max Parents configurations. Both base variants for SNP and MNP remain relatively stable around 5 units, regardless of the parent count. However, in the hybrid variants, latency increases significantly. For $MNPHybrid$, latency consistently exceeds 14 units across all configurations, which is roughly three times higher than the base variant. A similar pattern is visible for $SNPHybrid$, albeit at a slightly lower level (around 10–11 units). This shows that the added coordination from push-pull logic increases processing time per event.

Result Our hypothesis that MNP leads to higher latency compared to SNP is partially confirmed. While both strategies exhibit a similar latency increase as network size grows (Figure 27), the differences become more pronounced when increasing the graph density through the *Max Parents* parameter (Figure 28). Here, MNP consistently incurs higher latency than SNP after applying PrePP. This indicates that while the multi-node strategy improves transmission costs, the added overhead from coordination and inter-node communication—particularly under the push-pull mechanism—introduces additional latency. Thus, the trade-off between communication cost savings and real-time responsiveness must be considered when applying MNP in latency-sensitive scenarios.

6 Related Work

The combination of the two common decentralization strategies of CEP-operator placement and push-pull communication holding significant potential for enhancing performance in distributed networks, which is studied in several researches. In this section we present `Comet` [10] a complex event processing strategy in a distributed environment, `NeuroFlinkCEP` [22], a neurosymbolic complex event recognition (CER), optimized across IoT platforms, and the differentiation of these contributions to our work.

Comet: Complex Event Detection in Mobile Delay Tolerant Networks

One competing solution corresponding to our research challenge is Comet [10]. It focuses on decentralized complex event detection (CED) in mobile delay tolerant networks (DTNs) [10]. DTNs, in particular, pose significant challenges due to their inherent characteristics such as long transmission delays, frequent disconnections, high error rates, and elevated communication costs. Comet addresses these issues by employing a decentralized strategy to enhance the robustness and scalability of CED in such networks.

The CED process is distributed across multiple DTN nodes using a hierarchical CED tree in which intermediate nodes summarize events to reduce communication overhead. Comet leverages Dijkstra's algorithm, aiming to find the shortest path and with an h-function to optimize cost and delay in CED tree construction [10, 21]. Junction nodes host sub-CED processes that enable partial event detection closer to the data sources. In contrast to INES*, which employs multi-sink placement, Comet relies on single-sink placement with two-stage push-pull conversion algorithm. This algorithm works as follows: First, as many proactive push operations as possible will be converted into single-target pulls. Next, many of the remaining pushes will be converted into multi-target pulls [10, 21]. Comet's two-stage conversion heuristic could inspire enhancements to INES*'s PrePP component for additional cost reduction in scenarios with constrained links.

The Experimental Evaluation demonstrates the benefits of Comet compared to the centralized CED techniques. This approach achieves a cost reduction by up to 89% compared to the all push and around 56% when compared to the centralized optimal approach [10].

NeuroFlinkCEP: Neurosymbolic Complex Event Recognition Optimized across IoT Platforms

The NeuroFlinkCEP framework combines symbolic and neural Complex Event Recognition (CER) by extending Apache Flink with neurosymbolic capabilities for distributed IoT environments. CER patterns are expressed using regular expressions and are automatically translated into executable FlinkCEP jobs [22].

A key feature of the framework is the integration of domain-specific neural models for Simple Event (SE) detection. These models are embedded into `synapSEflow` operators and are executed locally on edge devices or cloud nodes depending on resource availability and event characteristics. The neural output is then consumed by FlinkCEP for symbolic pattern evaluation [22].

To optimize execution, NeuroFlinkCEP introduces the **DAG*4CER Optimizer**, an enhancement of the DAG algorithm, which incorporates CER-specific transformations:

Pattern Decomposition: Breaking down complex event patterns into smaller sub-patterns for distributed processing

Early Filtering and Predicate Pushing: Reducing the event load by filtering irrelevant events as early as possible

Reordering: Adjusting the evaluation order of events to improve matching efficiency

These optimizations aim to reduce latency, network load, and computational overhead by strategically deploying pattern detection close to the data source. The optimizer generates a physical execution plan based on a logical workflow, which can be designed in tools such as RapidMiner [22].

In contrast to INES*, which focuses on multi-node placement and push-pull optimization in hierarchical fog-cloud networks, NeuroFlinkCEP emphasizes hybrid symbolic-neural processing and high-level workflow optimization. Nonetheless, certain DAG*4CER strategies, particularly early filtering and pattern decomposition, could be adapted to INES* to improve operator partitioning and preprocessing before placement.

7 Conclusion

As the data volume grows significantly due to the increased number of IoT devices, efficient data processing in distributed environments becomes a key challenge. Especially in hierarchical fog-cloud architectures, SPSs must handle large-scale, distributed workloads while minimizing network overhead. In this thesis, we addressed these challenges by introducing INES*, an extension of the existing INES framework. Our enhancements includes MNP to improve scalability, distribution and transmission costs, and a graph compression technique to reduce the overall computation time for CEP workloads.

Methodology and Contribution. To reach our goal, we made the following adjustments:

- an *adaptation to hierarchical fog-cloud environments* by extending the flat placement logic of MNP, which originally only contains source nodes, to work efficiently in hierarchical topologies, taking into account the heterogeneity of fog and cloud nodes.
- a *integration of MNP*, which now allows interaction between MNP and SNP strategies, allowing flexible operator allocation depending on network conditions (Section 4.2.1).
- a *node characteristic hardware-constrain* by considering the computational power and reducing it, if possible placement is set. This helped to prevent overloading nodes and leads to a more realistic simulation. It increases transmission costs, leading potentially to an overhead (see Section 4.2.2).
- a *simple graph compression technique*, that filters irrelevant nodes from placement consideration, and reduces the required overall computation time(see Section 4.2.3).

All implemented components are publicly available as open-source to enable further research and development [1].

[1]`https://github.com/ND-2002/INES`

Findings and Results. Our evaluation across a wide range of experiments with varying graph density, event skew, and node-event ratio shows that INES* reduces transmission costs compared to INES with SNP only by up to **37%** in the tested configurations. The MNP strategy, especially when combined with PrePP, consistently outperforms INES in terms of network efficiency. While the maximum observed reduction is 37%, this value is influenced by the experimental setup of the used simulation environment. The simulated topologies distribute event types evenly across the network without regional clustering, which limits the potential for further data-locality optimization. Under regional clustering, MNP's advantage should be substantially larger, as hierarchical fog–cloud clustering minimizes cloud traversals and transmission costs by performing filtering and aggregation close to the sources. The achieved transmission cost reduction comes at the expense of increased **network latency** (transmission delays), particularly in *hybrid setups* (combination of OP and PP)—topologies that combine multiple fog layers with a cloud layer—under *large-scale* conditions (up to **300 nodes** in our experiments). In these scenarios, delays increased by approximately **20–35%** compared to SNP. Regarding computation time, MNP introduces a runtime overhead compared to SNP due to the higher complexity of multi-node decision-making. This overhead becomes more pronounced with network size, reaching its maximum at our evaluated maximal range of 300 nodes. At 300 nodes, the average time under MNP is about 45 seconds higher than SNP (160%). The simplification-based graph compression mitigates part of this effect:

- For SNP, compression only yields minor benefits, becoming visible from 200 nodes onward with around 3%.
- For MNP, compression reduces total execution time by up to **14%** at 300 nodes.

Although the runtime savings from compression are modest, they come without additional transmission cost penalties, making them a low-risk optimization. Despite these trade-offs, the scalability and communication benefits of MNP—especially in combination with PrePP—provide a strong argument for distributed placement strategies in future CEP fog-cloud environments.

Limitations. While the proposed enhancements improve performance in many scenarios, several limitations remain. After implementing MNP, the unchanged PrePP approach causes a significant latency overhead, especially in deep hierarchies or large networks. Another observation is that SNP still performs better in highly connected topologies or lower amount of event types, which highlights the need to further optimize the MNP strategy. Finally, while the simplified graph compression reduces computation time, further optimization is required for real-time deployment. We assumed that reducing the spectrum of possible target nodes would reduce the number of iterations (which proofed correct) and thus significantly improve computation time. However, since the networks become increasingly dense towards the root in terms of the number of events per node, our concept is limited.

Lessons Learned and Future Work. One interesting point we found out is, that the high impact of the resource-constrained is often neglected. It reduced the optimization potential, as node limitations restrict flexible placement. In the context of future work, several directions can be pursued to further enhance the capabilities and applicability of INES*. First, while the MNP strategy yield strong results in flat topologies, its performance in hierarchical fog-cloud environments can be improved. This includes refining the placement logic to better adapt to the characteristics of our fog-cloud environment. Second, the resource constraint model could be made more realistic by incorporating memory limitations or other node-specific metrics. Third, although our graph compression technique helped reduce computation time, the evaluation showed that the introduction of MNP itself increased overall runtime. Therefore, future work should explore code-level optimizations and refactoring of the placement logic to improve execution speed. Additionally, implementing alternative graph compression techniques, such as supernode-based compression (supernodes and superedges, see 2.4.1) could further reduce placement complexity and make the approach more scalable for large query workloads. Another point is PP, which only looks at the top placement of MNP and increases our network latency. To overcome this, an adaption to MNP is required.

Summary. To summarise, our work shows the positive effects of extending the INES framework with MNP and the graph compression technique on the transmission costs by varying configurations in a fog-cloud environment. Nevertheless, further optimization, particularly in handling latency, resource constraints, and real-time performance is necessary to make INES* viable for deployment in real-world, distributed CEP scenarios.

Bibliography

[1] Network Basics in Cloud Computing, pp. 145–195. Springer Nature Singapore, Singapore (2023), `https://doi.org/10.1007/978-981-19-3026-3_4`

[2] Ahammad, I.: Fog computing complete review: Concepts, trends, architectures, technologies, simulators, security issues, applications, and open research fields. SN Computer Science 4(765) (2023), `https://doi.org/10.1007/s42979-023-02235-9`

[3] Akdere, M., Çetintemel, U., Tatbul, N.: Plan-based complex event detection across distributed sources. Proceedings of the VLDB Endowment (PVLDB) 1(1), 66–77 (August 2008), `https://doi.org/10.14778/1453856.1453869`

[4] Akili, S.: On the need for distributed complex event processing with multiple sinks. In: Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems. p. 248–249. DEBS '19, Association for Computing Machinery, New York, NY, USA (2019), `https://doi.org/10.1145/3328905.3332520`

[5] Akili, S., Purtzel, S., Weidlich, M.: Inev: In-network evaluation for event stream processing. Proc. ACM Manag. Data 1(1) (May 2023), `https://doi.org/10.1145/3588955`

[6] Akili, S., Weidlich, M.: Muse graphs for flexible distribution of event stream processing in networks. p. 10–22. SIGMOD '21, Association for Computing Machinery, New York, NY, USA (2021), `https://doi.org/10.1145/3448016.3457318`

[7] Akili, S., Weidlich, M., Schlingloff, H., Penczek, W.: Reasoning on the efficiency of distributed complex event processing. Fundam. Inf. 179(2), 113–134 (Jan 2021), `https://doi.org/10.3233/FI-2021-2017`

[8] Alli, A.A., Alam, M.M.: The fog cloud of things: A survey on concepts, architecture, standards, tools, and applications. Internet of Things 9, 100177 (2020), `https://www.sciencedirect.com/science/article/pii/S2542660520300172`

[9] Atlam, H.F., Walters, R.J., Wills, G.B.: Fog computing and the internet of things: A review. Big Data and Cognitive Computing 2(2) (2018), https://www.mdpi.com/2504-2289/2/2/10

[10] Chen, J., Ramaswamy, L., Lowenthal, D.K., Kalyanaraman, S.: Comet: Decentralized complex event detection in mobile delay tolerant networks. In: 2012 IEEE 13th International Conference on Mobile Data Management. pp. 131–136 (2012)

[11] Cugola, G., Margara, A.: Deployment strategies for distributed complex event processing. Computing 95(2), 129–156 (2013), https://doi.org/10.1007/s00607-012-0217-9

[12] Cybenko, G., Brewington, B.: The Foundations of Information Push and Pull, pp. 9–30. Springer New York, New York, NY (1999), https://doi.org/10.1007/978-1-4612-1524-0_2

[13] Digitale, J.C., Martin, J.N., Glymour, M.M.: Tutorial on directed acyclic graphs. Journal of Clinical Epidemiology 142, 264–267 (2022), https://www.sciencedirect.com/science/article/pii/S0895435621002407

[14] Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M.: Network-wide complex event processing over geographically distributed data sources. Information Systems 88, 101442 (February 2020), https://doi.org/10.1016/j.is.2019.101442

[15] Giatrakos, N., Alevizos, E., Artikis, A., Deligiannakis, A., Garofalakis, M.N.: Complex event recognition in the big data era: a survey. VLDB J. 29(1), 313–352 (2020), https://doi.org/10.1007/s00778-019-00557-w

[16] He, Z., Peng, L.: Evaluation of fog topologies in fog planning for iot task scheduling. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. p. 2177–2180. SAC '20, Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3341105.3374027

[17] Kolchinsky, I., Schuster, A.: Real-time multi-pattern detection over event streams. In: Proceedings of the 2019 International Conference on Management of Data. p. 589–606. SIGMOD '19, Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3299869.3319869

[18] Li, C.T., Lin, S.D.: Egocentric information abstraction for heterogeneous social networks. In: 2009 International Conference on Advances in Social Network Analysis and Mining. pp. 255–260 (2009)

[19] Maccioni, A., Abadi, D.J.: Scalable pattern matching over compressed graphs via dedensification. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1755–1764. KDD '16, Association for Computing Machinery, New York, NY, USA (2016), `https://doi.org/10.1145/2939672.2939856`

[20] Majeed, A., Rauf, I.: Graph theory: A comprehensive survey about graph theory applications in computer science and social networks. Inventions 5(1) (2020), `https://www.mdpi.com/2411-5134/5/1/10`

[21] Meran, P.: CEP Strategies (November 2024), `https://github.com/Philippmer/INES/blob/master/docs/Bachelorarbeit_CEP_Strategies.pdf`, bachelor Thesis

[22] Ntouni, O., Banelas, D., Giatrakos, N.: Neuroflinkcep: Neurosymbolic complex event recognition optimized across iot platforms. PVLDB 14(1) (2020), `https://dl.acm.org/doi/pdf/10.1145/2020408.2020566`, demo Paper

[23] Patel, R., Prasad, L., Tandon, R., Rathore, N.P.S.: A Comprehensive Review on Edge Computing, Applications & Challenges, pp. 1–33. Springer International Publishing, Cham (2023), `https://doi.org/10.1007/978-3-031-28150-1_1`

[24] Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., Seltzer, M.: Network-aware operator placement for stream-processing systems. In: 22nd International Conference on Data Engineering (ICDE'06). pp. 49–49 (2006)

[25] Purtzel, S., Akili, S., Weidlich, M.: Predicate-based push-pull communication for distributed cep. In: Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems. p. 31–42. DEBS '22, Association for Computing Machinery, New York, NY, USA (2022), `https://doi.org/10.1145/3524860.3539640`

[26] Riondato, M., Upfal, E.: Graph summarization methods and applications: A survey. In: Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM). pp. 1661–1662. ACM (2011)

[27] Shen, Z., Ma, K.L., Eliassi-Rad, T.: Visual analysis of large heterogeneous social networks by semantic and structural abstraction. IEEE Transactions on Visualization and Computer Graphics 12(6), 1427–1439 (2006)

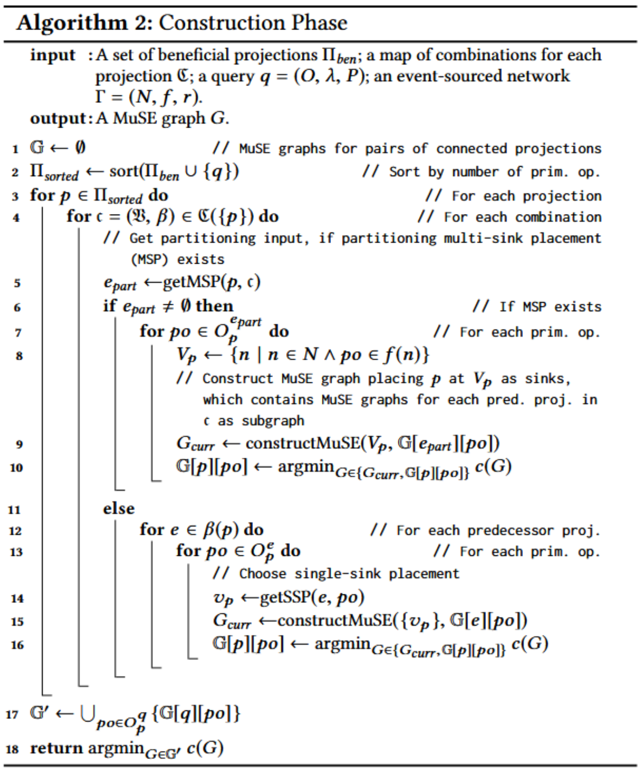
[28] Starks, F., Goebel, V., Kristiansen, S., Plagemann, T.: Mobile Distributed Complex Event Processing—Ubi Sumus? Quo Vadimus?, pp. 147–180. Springer International Publishing, Cham (2018), `https://doi.org/10.1007/978-3-319-67925-9_7`

[29] Toivonen, H., Zhou, F., Hartikainen, A., Hinkka, A.: Compression of weighted graphs. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 965–973. KDD '11, Association for Computing Machinery, New York, NY, USA (2011), `https://doi.org/10.1145/2020408.2020566`

[30] Zeuch, S., Chaudhary, A., Monte, B.D., Gavriilidis, H., Giouroukis, D., Grulich, P.M., Breß, S., Traub, J., Markl, V.: The nebulastream platform for data and application management in the internet of things. In: 10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings (2020), `http://cidrdb.org/cidr2020/papers/p7-zeuch-cidr20.pdf`, available at `http://cidrdb.org/cidr2020/papers/p7-zeuch-cidr20.pdf`

[31] Ziehn, A.: Complex event processing for the internet of things. In: Proceedings of the VLDB 2020 PhD Workshop co-located with the 46th International Conference on Very Large Databases (VLDB). vol. 2652. CEUR-WS.org (2020), `https://ceur-ws.org/Vol-2652/paper01.pdf`

Appendix A. Further Details on the Solution Approach



Algorithm 2: Construction Phase

```
input : A set of beneficial projections $\Pi_{ben}$; a map of combinations for each
        projection $\mathfrak{C}$; a query $q = (O, \lambda, P)$; an event-sourced network
        $\Gamma = (N, f, r)$.
output: A MuSE graph $G$.
$\mathbb{G} \leftarrow \emptyset$                    // MuSE graphs for pairs of connected projections
$\Pi_{sorted} \leftarrow$ sort($\Pi_{ben} \cup \{q\}$)          // Sort by number of prim. op.
for $p \in \Pi_{sorted}$ do                                    // For each projection
    for $\mathfrak{c} = (\mathfrak{B}, \beta) \in \mathfrak{C}(\{p\})$ do                // For each combination
        // Get partitioning input, if partitioning multi-sink placement
           (MSP) exists
        $e_{part} \leftarrow$getMSP($p$, $\mathfrak{c}$)
        if $e_{part} \neq \emptyset$ then                               // If MSP exists
            for $po \in O_p^{e_{part}}$ do                          // For each prim. op.
                $V_p \leftarrow \{n \mid n \in N \wedge po \in f(n)\}$
                // Construct MuSE graph placing $p$ at $V_p$ as sinks,
                   which contains MuSE graphs for each pred. proj. in
                   $\mathfrak{c}$ as subgraph
                $G_{curr} \leftarrow$ constructMuSE($V_p$, $\mathbb{G}[e_{part}][po]$)
                $\mathbb{G}[p][po] \leftarrow \text{argmin}_{G \in \{G_{curr}, \mathbb{G}[p][po]\}}\ c(G)$
        else
            for $e \in \beta(p)$ do                       // For each predecessor proj.
                for $po \in O_p^e$ do                            // For each prim. op.
                    // Choose single-sink placement
                    $v_p \leftarrow$getSSP($e$, $po$)
                    $G_{curr} \leftarrow$constructMuSE($\{v_p\}$, $\mathbb{G}[e][po]$)
                    $\mathbb{G}[p][po] \leftarrow \text{argmin}_{G \in \{G_{curr}, \mathbb{G}[p][po]\}}\ c(G)$
$\mathbb{G}' \leftarrow \bigcup_{po \in O_p^q} \{\mathbb{G}[q][po]\}$
return $\text{argmin}_{G \in \mathbb{G}'}\ c(G)$
```

Figure 29: Algorithm of the Construction Phase.
Source: [6]

Appendix B. Extended Version of the Experimental Results

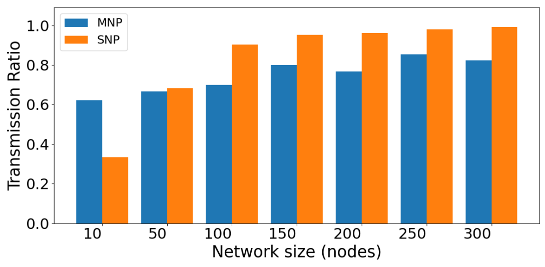


Figure 30: High Query Complexity under increasing Network Size