Technische Universität Berlin

Chair of Database Systems and Information Management

Bachelor's Thesis

CEP Strategies in Fog Cloud Environments

Philipp Emanuel Meran
Degree Program: Business Informatics
Matriculation Number: 409792

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

Advisor
Ariane Ziehn

Submission Date
29.11.2024

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

Berlin, 29.11.2024

. . . .

Philipp, Emanuel Meran

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit eigenständig ohne Hilfe Dritter und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe. Alle Stellen die den benutzten Quellen und Hilfsmitteln unverändert oder sinngemäß entnommen sind, habe ich als solche kenntlich gemacht.

Sofern generative KI-Tools verwendet wurden, habe ich Produktnamen, Hersteller, die jeweils verwendete Softwareversion und die jeweiligen Einsatzzwecke (z.B. sprachliche Überprüfung und Verbesserung der Texte, systematische Recherche) benannt. Ich verantworte die Auswahl, die Übernahme und sämtliche Ergebnisse des von mir verwendeten KI-generierten Outputs vollumfänglich selbst.

Die Satzung zur Sicherung guter wissenschaftlicher Praxis an der TU Berlin vom 15. Februar 2023. https://www.static.tu.berlin/fileadmin/www/10002457/K3-AMBl/Amtsblatt_2023/Amtliches_Mitteilungsblatt_Nr._16_vom_30.05.2023.pdf habe ich zur Kenntnis genommen.

Ich erkläre weiterhin, dass ich die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Berlin, 29.11.2024

. . . .

Philipp, Emanuel Meran

Chap

Die rasante Zunahme der von IoT-Geräten generierten Daten bringt die Netzwerkkapazitäten an ihre Grenzen. Das Fog-Cloud-Paradigma begegnet dieser Herausforderung, indem es die Datenverarbeitung näher an der Quelle der Generierung ermöglicht. CEP-Systeme, die für die Analyse von IoT-Datenströmen unerlässlich sind, stoßen jedoch an ihre Grenzen, wenn wir bestehende Algorithmen direkt und ohne Anpassung auf neuartige Fog-Cloud-Umgebungen anwenden. Zu den wichtigsten CEP-Strategien gehören die Operator-Platzierung, bei der die optimalen Knoten für Rechenaufgaben bestimmt, und Push-Pull, das Kommunikationsprotokolle zur Reduzierung der Netzwerknutzung nutzt, erfordern eine Modifizierung, um die Fog-Cloud-Ressourcen voll auszuschöpfen.

In dieser Arbeit untersuchen wir, wie CEP-Strategien an die einzigartigen Eigenschaften von Fog-Cloud-Topologien angepasst werden können, um eine effiziente und skalierbare verteilte Datenverarbeitung zu ermöglichen. Um modernste Operator-Platzierungs- und Push-Pull-Algorithmen zu kombinieren und anzupassen, stellen wir INES vor, ein einheitliches System für Fog-Cloud-Umgebungen, das moderne CEP-Techniken in eine einheitliche Architektur integriert. Um die Notwendigkeit der Anpassung von CEP zu demonstrieren, führen wir umfangreiche Experimente in simulierten Umgebungen durch und bewerten INES, wobei wir die wichtigsten Eigenschaften der angestrebten Fog-Cloud-Umgebung untersuchen. Unsere Bewertung zeigt, dass INES eine Reduzierung der Übertragungskosten um bis zu 90% durch netzinterne Vorverarbeitung und Kommunikationsoptimierung erreicht und damit traditionelle zentralisierte Ansätze deutlich übertrifft. Diese Ergebnisse unterstreichen die entscheidende Rolle der Interkonnektivität bei der Maximierung von Übertragungseinsparungen, da sie die Leistung von Betreiberplatzierungs- und Push-Pull-Kommunikationsstrategien

Abstract

The rapid growth of data generated by IoT devices pushes network capabilities to their limits. The fog-cloud paradigm addresses this challenge by enabling data processing closer to the source of generation. However, CEP systems, essential for analyzing IoT data streams, face limitations when we apply existing algorithms directly to novel fog-cloud environments without adaptation. Prominent CEP strategies are operator placement, which determines the optimal nodes for computing tasks, and push-pull, which utilizes communication protocols to reduce network usage, require modification to fully leverage fog-cloud resources.

In this work, we explore how CEP strategies can be adapted to the unique characteristics of fog-cloud topologies to enable efficient and scalable distributed data processing. To combine and adapt state-of-the-art operator placement and push-pull algorithms, we propose INES, a unified system for fog-cloud environments integrating modern CEP techniques into a cohesive architecture. To demonstrate the necessity of adapting CEP, we conduct extensive experiments in simulated environments and evaluate INES, investigating key properties of the targeted fog-cloud setting. Our evaluation shows that INES achieves up to 90% reductions in transmission costs through in-network pre-processing and communication optimizations, significantly outperforming traditional centralized approaches. These results underscore the critical role of interconnectivity in maximizing transmission savings, as it directly influences the performance of operator placement and push-pull communication strategies.

Contents

List of Figures

List of Tables

List of Abbreviations

DIMA	Database Systems and Information Management
PrePP	Predicate Based Push-Pull
CEP	Complex Event Processing
INEv	In-Network Evaluation
IoT	Internet of Things
DAG	Directed Acyclic Graph

List of Algorithms

1 Introduction

As the Internet of Things (IoT) continues to expand, it is revolutionizing industries such as healthcare, energy, and finance by enabling innovative data management applications [19]. These applications process enormous amounts of data generated by geographically distributed devices [1, 10]. This exponential increase in data volume poses significant challenges for traditional data processing methods. This work focuses on the stream processing paradigm, Complex Event Processing (CEP), which is essential for applications like healthcare surveillance. CEP systems detect complex patterns in event streams and respond automatically [30]. For example, a CEP engine in a health monitoring system could identify critical conditions, such as simultaneous spikes in heart rate and blood pressure, and alert medical personnel.

Traditionally, cloud computing has been the primary solution for handling large-scale data processing tasks, where all events and data streams are transferred to centralized cloud servers. Although the cloud offers significant computing and storage resources, its exclusive use is becoming increasingly insufficient due to network congestion and excessive bandwidth consumption caused by the increasing number of IoT devices [1, 32]. To address these limitations, decentralized concepts have gained prominence by distributing computing tasks across multiple nodes in the network, reducing dependency on a central server [1, 5]. However, purely decentralized approaches lack the robust resources of the cloud, such as virtually unlimited computing power and storage capacity. This lack limits their scalability and ability to handle complex processing tasks requiring significant resources [1].

As a proposed solution, the fog cloud environment, which represents a tree-like network topology, combines the strengths of cloud computing and fog computing. This unified environment brings computing and storage resources closer to data generation sources while leveraging the capabilities of the cloud. Processing and reducing data at the edge of the network reduces latency and facilitates real-time analysis [1]. However, while cloud optimization serves as a fallback in fog-cloud environments, fully leveraging the benefits of the attached fog layer requires adapting state-of-the-art strategies for workload distribution and network communication.

We modify state-of-the-art strategies in decentralized CEP for fog-cloud environments. This includes optimizing in-network processing to reduce transmission costs and exploring **operator placement**, which allocates tasks across the network to minimize latency, and **push-pull communication**, which dynamically decides whether data is pushed or pulled based on workload and network conditions.

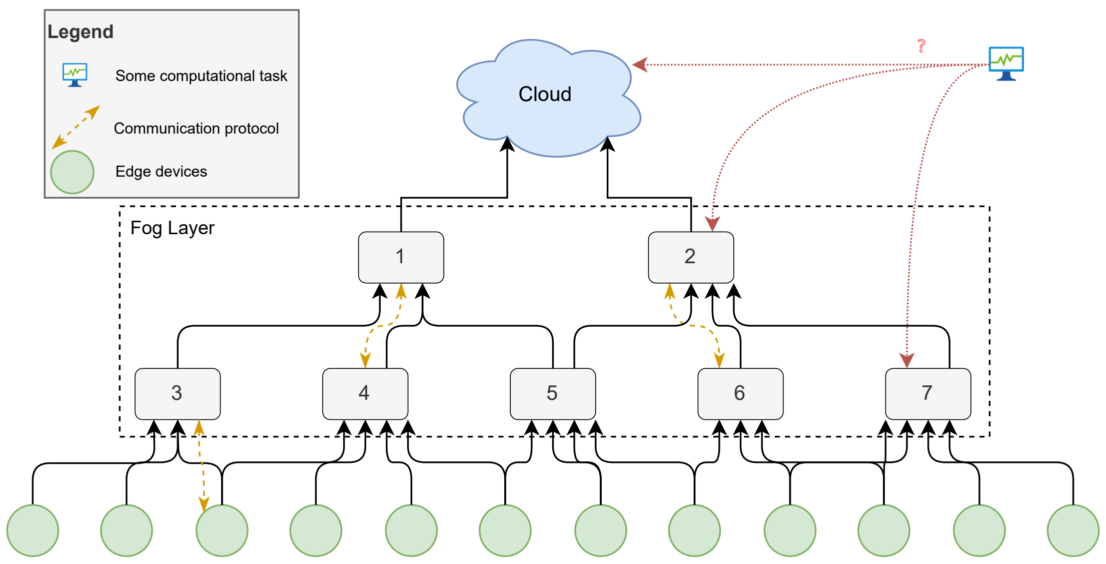


Figure 1: Fog-cloud network

Figure 1 illustrates a fog-cloud topology. It highlights two primary challenges: determining the optimal placement of computational tasks among the available nodes and deciding on the most effective communication protocols—specifically, whether to employ push or pull mechanisms.

Novelty

Although research has been conducted on CEP systems in distributed settings, many studies do not consider the hierarchical network structure characteristic of fog-cloud topologies [3, 4, 27]. Leading to suboptimal utilization of resources and higher transmission costs. Furthermore, it has been found that combining operator placement and push-pull increases the performance of CEP systems [7, 10].

However, this work does not consider fog-cloud computing and does not employ today's state-of-the-art algorithms, i.e., In-Network Evaluation (INEv) and Predicate-based Push-Pull (PrePP) [3, 10, 27]. INEv implements an operator placement strategy and enhances CEP performance by allowing event processing to occur directly within the network nodes, reducing the need to transmit all raw data to a central processor. PrePP, on the other hand, represents a push-pull mechanism, which improves efficiency by dynamically deciding whether to push or pull data based on contextual information, optimizing bandwidth usage.

Nebula Stream [32], a representative system for fog-cloud computing, exemplifies the potential of combining fog and cloud resources to handle distributed event processing in hierarchical environments. Building on this concept, we propose INES (In-Network-Evaluation for Nebula Stream), an in-network algorithm that combines state-of-the-art decentralized distribution strategies, i.e., INEv [3] and PrePP [27] in a single algorithm tailored to the characteristics of fog-cloud environments. By simulating the novel unified fog-cloud environment, we aim to demonstrate how INES can enhance CEP system performance and reduce transmissions in a network.

Research Questions

To reach our goal and investigate how state-of-the-art algorithms perform in novel fog-cloud environments, we aim to answer the following research questions:

1. How do state-of-the-art approaches perform in fog-cloud environments?
2. In a fog-cloud environment, which parameters affect INES the most?
3. How scalable is our solution?

To answer these questions, we simulate integrating operator placement strategies and push-pull communication protocols using modern algorithms such as INEv [3] and PrePP [27] within a fog cloud network topology. By analyzing the performance of these combined strategies, we aim to identify key parameters that influence the benefit of decentrlaized strategies.

Anticipated Impact

This research provides a foundation for evaluating algorithms in fog-cloud CEP systems, enabling informed decisions about employing operator placement and push-pull communication to reduce transmissions under specific conditions. The findings guide further simulations, real-world testing, and future developments for systems like NES [32].

2 Scientific Background

This chapter provides the foundational concepts for this work. Section 2.1 introduces network environment types, while Section 2.2 presents two approaches, operator placement and push-pull, for optimizing CEP in distributed networks. Finally, Section 2.3 details our network model and query language.

2.1 Network Environments

Networks can be effectively modeled using graph theory [28], where nodes represent computing devices, and edges represent communication links between them.

In the following sections, we describe three types of computing environments—cloud, fog, and fog-cloud—by detailing the kind of graph representing each network and then discussing their characteristics and limitations.

2.1.1 Cloud Environments

Network Graph Structure

In cloud computing, the network can be represented as a *star topology*, a type of undirected graph where multiple peripheral nodes (clients or edge devices) are connected to a central node (cloud data center) but not to each other [16, 33]. This centralization emphasizes the primary role of the cloud server in data processing and storage.

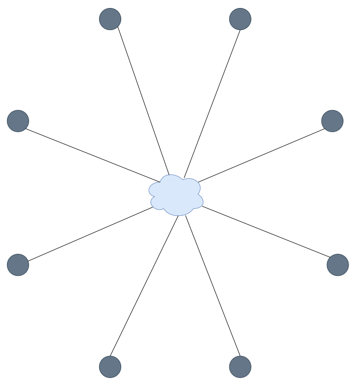


Figure 2: Star network graph

We visualize such a star graph in Figure 2, in which we have 8 nodes, all connected to a central node in the center. The central node represents the cloud.

Characteristics

Cloud computing has emerged as a transformative paradigm in distributed computing, offering on-demand access to a shared pool of configurable computing resources over the internet [11, 22, 26]. Services are delivered over networks and accessed through standardized endpoints and protocols [22]. A key feature is its *elasticity*; as demand increases, computing capabilities can be scaled up dynamically, giving consumers the impression of virtually unlimited resources [15].

Limitations

The centralized nature of the cloud introduces certain limitations. Latency issues arise due to the physical distance between data sources (edge devices) and cloud servers, which can negatively impact the performance of latency-sensitive applications [1]. The star topology means all data must pass through the central node, creating potential bottlenecks. Additionally, transmitting large volumes of data to and from the cloud can be bandwidth-intensive and costly [3, 18, 25, 26].

2.1.2 Fog Environments

Network Graph Structure

Fog or edge computing environments are represented by *undirected mesh graphs*, where nodes are decentralized without a central coordinating node [13, 14]. In some cases, the graph may be fully connected (complete graph), allowing direct communication between any pair of nodes. However, scalability considerations make an incomplete mesh topology more common in practical scenarios.

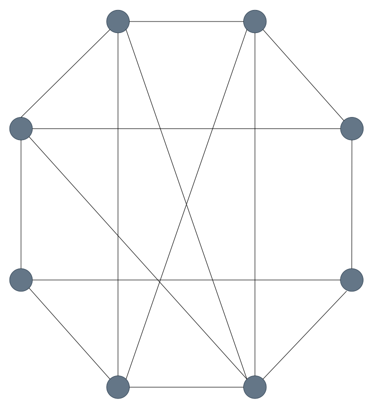


Figure 3: Incomplete mesh network graph

Figure 3 shows an incomplete mesh graph that illustrates the decentralized nature of the fog topology. Not all nodes are directly connected to each other, but every node can be reached via the decentralized network structure.

Characteristics

Fog computing relies on a distributed approach that makes targeted use of the computing and storage capacities of the devices in the network [1, 5, 6]. Since data processing takes place locally or in close proximity to the source, both latency and bandwidth consumption are reduced. This proximity is crucial for applications that depend on real-time data, such as autonomous vehicles, smart grids, or industrial automation [1, 20]. The main advantage of a fog environment is to bring computing resources closer to the network periphery where IoT devices operate, thereby reducing latency and bandwidth usage. The mesh topology enables direct communication between nodes, which increases responsiveness and supports real-time processing.

Limitations

While decentralized networks minimize latency, individual edge devices often have limited computing and storage capabilities compared to centralized cloud servers [1]. Complex tasks may therefore require more time to process [26].

2.1.3 Fog-Cloud Environments

Network Graph Structure

Fog-cloud environments combine cloud and fog computing elements, resulting in a hierarchical network graph modeled as an *N-tier hierarchical directed acyclic graph (DAG)*. This graph consists of multiple layers: the cloud layer at the top, one or more fog layers in the middle, and edge devices (IoT devices) at the bottom [1, 20]. Edges represent communication paths, typically directed from lower to higher layers, facilitating structured data flow.

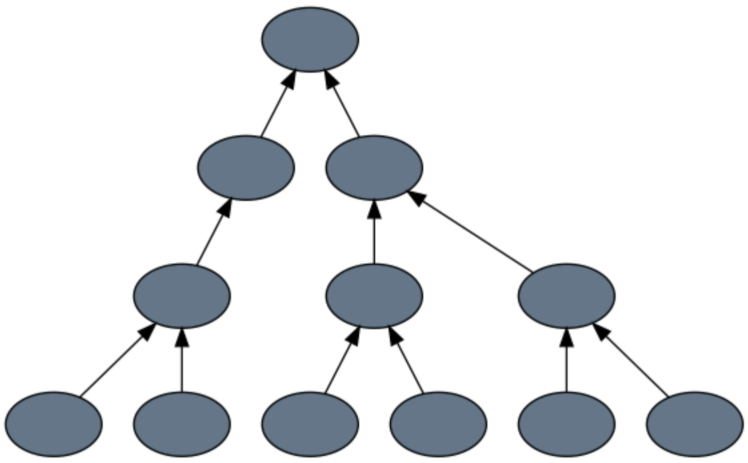


Figure 4: Directed acyclic network graph

Characteristics

This hierarchical structure allows for processing tasks to be distributed across different layers. The fog layers act as intermediaries that process data closer to its source. Fog-cloud computing extends cloud computing capabilities by introducing a decentralized intermediary layer that provides computing, connectivity, and storage services close to IoT devices [1, 5, 20]. The system improves efficiency, scalability, and responsiveness by offloading tasks from the cloud to the fog nodes. Combining both fog and cloud environments is beneficial for distributing workloads closer to the edge before sending the data to the cloud, thereby optimizing network traffic and enhancing overall system performance.

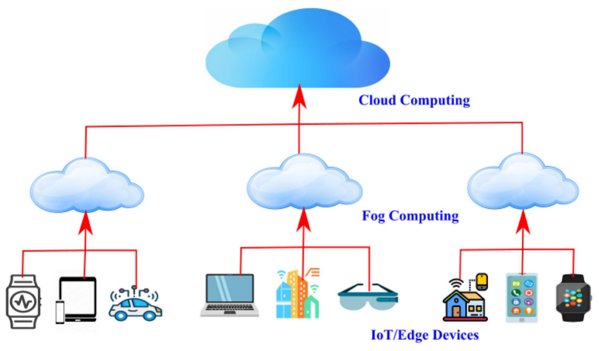


Figure 5: Architecture of the fog-cloud computing model.
Source: [1]

In Figure 5, we illustrate a fog-cloud architecture. At the bottom of this structure are data-generating nodes, or end devices, that produce large amounts of data. This data flows upward to the fog layer, undergoing preprocessing and other computations. Finally, only the relevant data is sent to the cloud for storage and further processing.

Limitations

The fog-cloud model introduces complexity in network management and orchestration. Seamless integration and communication across different layers can be challenging due to the heterogeneity of devices and protocols involved. Additionally, balancing the load between fog and cloud resources requires adapted algorithms to maximize efficiency [1].

In conclusion, the fog-cloud environment enables the use of advantages of both the cloud and fog computing topologies in one unified environment. The characteristics of all three environments can be summarized in the following table.

Attribute	**Cloud**	**Fog**	**Fog-Cloud**
Geo-distribution	Centralized	Decentralized	Both
Latency	High	Low	Low to high
Storage	Potentially unlimited	Finite	Potentially unlimited
Processing capability	Potentially unlimited	Finite	Potentially unlimited
Data analysis	Data at rest	Data in motion	Data in motion
Location awareness	No	Yes	Yes

Table 1: Comparison of cloud, fog, and fog-cloud attributes
Source: [23]

These characteristics of fog-cloud environments are emphasized in the open-source project NebulaStream [21], which focuses on stream processing for highly distributed, heterogeneous environments. NebulaStream supports diverse sensors within the cloud-edge continuum and operates on a hierarchical topology of heterogeneous devices, underscoring the potential of fog computing in managing complex, distributed data processing networks [21].

2.2 Distribution Strategies

In a CEP context, utilizing distribution strategies is crucial to optimizing performance. These strategies aim to optimize the distribution of data and processing tasks across multiple nodes. This enables CEP systems to handle high volumes of data streams and complex event patterns. In the subsequent sections, we elaborate on two strategies: operator placement and push-pull.

2.2.1 Operator Placement

In traditional CEP approaches, all data is sent to one single processing unit, often the cloud, where all data is evaluated and processed [1, 12]. While this centralized approach simplifies data management and processing, it presents significant limitations in scenarios involving the Internet of Things (IoT) and distributed networks. The exponential growth of IoT devices and the volume of data they generate necessitates more efficient data processing strategies to overcome issues such as network bandwidth limitations, latency, and scalability [29]. New network topologies and paradigms have emerged to address these challenges, emphasizing the decentralization of processing tasks. One such paradigm is **fog computing**, which involves moving computational capabilities closer to data sources. This shift aims to reduce latency, decrease bandwidth consumption, and enhance overall system responsiveness. However, it introduces a critical challenge: **determining the optimal placement of computing tasks (operators) within a distributed network** to ensure both effectiveness and efficiency [29]. The operator placement problem aims to solve this issue by computing the optimal node/placement to perform calculations in a distributed network. In the following sections, we define the problem and outline its formulation.

Problem Definition

The **Operator Placement Problem** is known to be **NP-hard** [3, 17, 29], involving the strategic assignment of computational tasks (operators) to nodes within a distributed system. The goal is to optimize certain performance metrics, such as minimizing communication costs, balancing load, or maximizing resource utilization while working under various constraints [17].

Key components involved in formulating the operator placement problem include:

- **Flow Graph**: Represents the logical flow of data through the system, consisting of data sources and a collection of tasks or operators that process the data. It models how data is transformed and propagated through various processing stages [17].
- **Topology:** Describes the physical layout of the network, including the nodes (processing units) and the connections between them. It defines how data can physically flow between nodes and the potential paths for communication [17].
- **Set of constraints:** Real-world scenarios impose various constraints on operator placement. These may include resource limitations (e.g., CPU, memory), privacy and security requirements, or policies that restrict the use of certain nodes [17].
- **Statistics:** Prior to solving the placement problem, certain statistical information is available, such as the message output rate of each operator, expected computational resource consumption, and network latency. This information is crucial for making informed placement decisions [17].

The primary objective is to assign operators to processing nodes in a manner that satisfies all constraints.

Problem Formulation

The operator placement problem involves assigning operators to nodes within a system while adhering to resource and operational constraints [29].

The constraints of the problem are as follows [24]:

$$\min_x F(x)$$

subject to:

1. **Resource Constraints**

$$\sum_{i \in V_{dsp}} C_i x_{i,u} \leq C_u \quad \forall u \in V_{res} \tag{1}$$

This constraint ensures that the cumulative resource consumption of operators assigned to a node u does not exceed the node's available resources C_u. Here, C_i represents the resource requirement of the operator i, V_{dsp} is the set of operators (processing tasks), and V_{res} is the set of available resources (nodes).

2. **Operator Assignment Constraints**:

$$\sum_{u \in V_{res}} x_{i,u} = 1 \quad \forall i \in V_{dsp} \tag{2}$$

   This constraint guarantees that each operator i is assigned to exactly one node. The variable $x_{i,u}$ is a binary variable indicating whether operator i is assigned to node u ($x_{i,u} = 1$) or not ($x_{i,u} = 0$).

3. **Binary Assignment Variables**:

$$x_{i,u} \in \{0, 1\} \quad \forall i \in V_{dsp}, u \in V_{res} \tag{3}$$

   This ensures that the assignment variables are binary, reflecting the discrete nature of the placement decisions.

In some scenarios, allowing operators to be placed on multiple nodes **(multi-node placement)** can be beneficial for load balancing and fault tolerance. However, this adds complexity to the problem and may require modifying constraints (2) and (3) to permit multiple assignments.

The operator placement problem is a fundamental challenge in designing efficient, scalable, and responsive distributed CEP systems, especially in the context of IoT. By formulating it as an optimization problem and employing various algorithmic strategies, effective solutions that enhance system performance while adhering to necessary constraints can be found.

Ongoing research continues to explore innovative methods for operator placement, addressing the complexities of dynamic environments and multi-objective optimization and contributing to the advancement of distributed processing technologies.

2.2.2 Push-Pull Communication

Communication strategies in distributed systems play a crucial role in optimizing data exchange between nodes. Communication in this context encompasses the methods and protocols that control data transfer between distributed nodes. Two fundamental paradigms are push and pull communication [8]. In the push paradigm, data producers proactively send information to consumers without an explicit request, as they anticipate the consumer's needs. In contrast, in the pull paradigm, the consumer requests the information from the producer when it is needed [2, 10].

In the context of Complex Event Processing (CEP), distributed nodes generate event streams that are processed to detect complex patterns or important events. Traditional CEP systems often rely on a push-based model in which all generated events are sent to central processing nodes [2, 10]. However, this approach can lead to network congestion and inefficiencies, especially at high event rates and when only a small fraction of events are relevant for query evaluation [10, 27].

A combination of push and pull communication strategies can help to significantly reduce unnecessary data transfers. This hybrid approach uses the characteristics of event rates and temporal relevance to optimize communication.

Push and Pull Communication Paradigms

- **Push Communication:** Producers send data to consumers as soon as it is available, without the need for an explicit request. This paradigm is well suited when consumers need data regularly or when immediate delivery is crucial [2, 10].

- **Pull Communication:** consumers request data from producers when they need it. This paradigm is efficient when the consumer only needs data occasionally or when the data production rate is high but only some of the data is relevant [2, 10].

Application in Complex Event Processing

In CEP systems, events are often time-sensitive and may have varying occurrence rates across different nodes. Efficient communication strategies must account for these variations to reduce network load without compromising the timeliness of event detection.

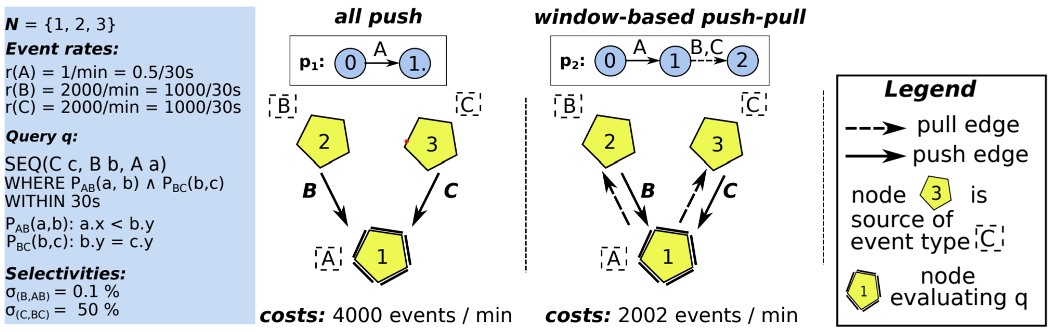


Figure 6: Push compared to push-pull communication
Source: [27]

Consider an illustrative example, see Figure 6 involving a network with three nodes:

- **Node 1**: Evaluates a specific query and generates **event A** at a low rate of **1 event per minute**.
- **Node 2**: Generates **event B** at a high rate of **2000 events per minute**.
- **Node 3**: Generates **event C** at a high rate of **2000 events per minute**.

In a conventional all-push model, nodes 2 and 3 continuously transmit their high-rate event streams to node 1. This results in 4000 events per minute being sent over the network, most of which may not contribute to the query evaluation involving event A [27].

Window-Based Push-Pull Communication Strategy

To enhance efficiency, a **window-based push-pull** communication strategy can be employed:

1. **Triggering Pull Requests**: Node 1 uses the occurrence of its low-rate event A as a trigger to initiate pull requests. Upon generating event A, node 1 sends pull requests to nodes 2 and 3.
2. **Temporal Event Selection**: Nodes 2 and 3, upon receiving a pull request, send back only the events generated within a specific **time window** relevant to the query (e.g., the last 30 seconds). This ensures that only events potentially matching with event A are transmitted.

Adopting this strategy significantly reduces the number of events transmitted from nodes 2 and 3. If each node generates **1000 events** within the 30-second window, only **2000 events** are sent in response to the pull requests, compared to the continuous transmission of 4000 events per minute in the all-push model. The transmission overhead introduced by the pull requests is minimal, resulting in an overall transmission reduction of nearly **50%** [27].

Benefits and Implications

The push-pull communication strategy offers several benefits:

- **Reduced Network Load**: By transmitting only relevant events, the strategy decreases network congestion and bandwidth usage.
- **Efficient Resource Utilization**: Processing nodes handle fewer events, leading to a lower computation load.
- **Scalability**: The system can better handle increases in event rates or the number of nodes without proportionally increasing resource requirements.

Integrating push and pull communication strategies in CEP systems can substantially optimize network transmissions. By intelligently orchestrating data exchange based on event relevance and timing, it is possible to enhance performance, reduce resource consumption, and maintain the responsiveness of event detection mechanisms [7, 10].

This hybrid communication model is a promising approach for managing large-scale, distributed CEP systems, particularly in environments with high event rates and strict resource constraints.

Pull communication refers to a node requesting information by sending a pull request to a node holding the desired information. Whereas push communication refers to the node sending information anticipating the need of the user to receive this information [2, 8]. In the CEP context, event streams, i.e., data being generated by distributed nodes, all events are sent throughout the network to be evaluated by certain nodes that are capable of running the query. Typically, in scenarios with huge amounts of data, only a small fraction of events are required to generate the matches. This results in an overflow of the network with data that is not relevant for the use case [27].

2.3 Preliminaries

2.3.1 Network Model

In this study, we consider a hierarchical network graph consisting of N nodes, modeled as a directed acyclic graph (DAG). The network is structured such that nodes are arranged in layers, with nodes at the lowest layer being the sole source nodes, i.e., event-generating nodes of simple events. Each node in the network can have parent and child nodes, and data flow is only possible in a directed manner, from child to parent, but we allow pull requests to be sent bi-directionally. The edges in the graph represent the communication paths. Data can travel within the network where we assume reliable communication links with minimal latency. Variations in network conditions, such as bandwidth limitations, could be considered in future work to simulate more realistic scenarios. Nodes within the network are heterogeneous, exhibiting varying levels of computing power and memory capacity. This heterogeneity is structured so nodes higher in the hierarchy possess greater computing and memory resources. The node at the topmost level of the hierarchy, simulating the cloud, is assumed to have virtually infinite computing and memory capacity, serving as a central processing unit for the network.

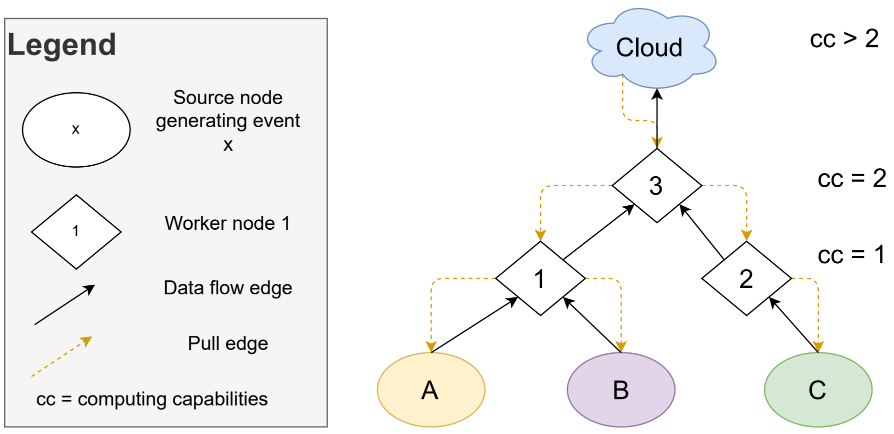


Figure 7: Hierarchical network graph
Source: [23]

We visualize a fog-cloud network topology as a DAG in Figure 7. In our topology, nodes are hardware devices in a network which are connected to other nodes via communication protocols. Our network model consists of 3 different types of nodes. First, the cloud representation, a node with virtually infinite computing and memory capabilities, is the topmost node in the hierarchy. Second, the fog nodes have reduced computational and memory abilities, depending on their location in the hierarchy. Lastly, the sensor or data-generating nodes at the bottom of the hierarchy have no computing capabilities.

In summary, this network model provides a robust framework for simulating complex event processing in fog cloud environments, accommodating the variability and scalability inherent in such systems.

2.3.2 Query Language

A query in Complex Event Processing (CEP) is a set of conditions that monitors data streams. It aims to detect patterns by matching specific sequences or conditions in events. When events in a stream satisfy a query's criteria, they create a match and generate a higher-level, complex event as output.

Multiple queries are often used together to create a query workload. This workload, a collection of queries working simultaneously, is structured as a forest, meaning a set of independent query trees [4]. Using multiple queries allows for a broader range of patterns to be detected across different data streams and is particularly useful for simulating complex, real-world scenarios. Each simulation run in this work will include a query workload to assess performance.

Queries contain two main types of operators:

1. **Primitve Operators** - Operate on a single event type, handling individual event conditions.

2. **Composite Operators** - Combine multiple event types, utilizing specific patterns or sequence rules.

In this work, we will mainly focus on two different composite operator types:

- **AND** creates a match if all its child operators generate a match

- **SEQ** similar to the AND operator, SEQ also considers the specified order of events in the sequence

As queries become more complex, they can be divided into subqueries and evaluated in different network parts for efficiency. Breaking down a query into smaller, manageable parts is called projection, where each subquery focuses on a subset of the event types and inherits predicates and time windows from the main query. To ensure correct evaluation, projections are combined into a structure represented as a directed acyclic graph (DAG), with each projection as a vertex. MuSE [4] and INEv [3] are both examples of implementing a query-splitting process and optimizing operator placement in a distributed CEP context.

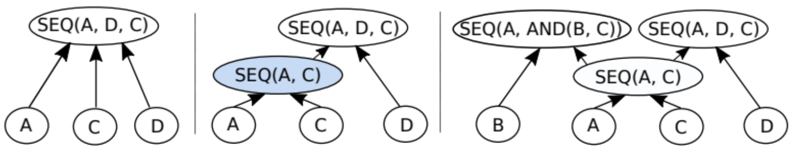


Figure 8: Query splitting mechanism
Source:[3]

Figure 8 illustrates how a query can be transformed into an operator tree. In this tree structure, each operator corresponds to a computational task that can be executed independently on different nodes, provided that the node has access to the required events.

In the middle operator tree depicted in the figure, the complex query *SEQ(A, D, C)* is split into two sub-queries: *SEQ(A, C)* and *SEQ(A, D, C)*. By processing *SEQ(A, C)* first, we can identify preliminary matches that are then forwarded to the *SEQ(A, D, C)* operator for further analysis. This approach ensures that only the relevant matches from *SEQ(A, C)* are transmitted to the top-level node, thereby reducing data transmission and enhancing the efficiency of distributed processing.

The projections are assigned to network nodes for evaluation, which can occur in two ways [3]:

- **Single-Node Placement**: All events are processed on a single node.
- **Multi-Node Placement**: Events are distributed across multiple nodes, with different parts of the query handled by different nodes.

Optimally splitting and placing subqueries across nodes is an **NP-Hard problem**, as discussed in the operator placement section [3].

3 Problem Definition

The INEv repository [3] serves as the basic framework for our project and provides algorithms for calculating optimal operator placements in undirected networks. However, the original design of INEv is based on assumptions that only partially match the realities of fog cloud environments. In particular, it assumes that the network is fully connected i.e., each node can communicate directly with any other node. This assumption ignores the topologies of the fog clouds hierarchical and constrained nature, where communication between nodes is restricted to predefined paths.

In fog cloud environments, the network topology has a hierarchical structure consisting of multiple layers, from edge devices (fog nodes) to centralized cloud servers. Communication is often constrained by network policies, security requirements, and physical limitations [5]. Therefore, the assumption of a fully connected network does not reflect the essential characteristics of these environments and can lead to suboptimal or impracticable operator placement strategies.

To address this issue, we introduced an **directed acyclic graph (DAG)** to model the hierarchical structure of fog-cloud networks. In this DAG representation, nodes are arranged in layers that reflect the hierarchy from fog nodes (closer to the edge) to the cloud (centralized, higher in the hierarchy). Each directed edge represents a unidirectional communication link and corresponds to the data flow in fog-cloud architectures [5]. This modeling decision ensures that the network topology correctly reflects the real-world constraints of fog-cloud environments, where data typically flows from sensors and edge devices up to centralized processing units.

However, modeling the network as a DAG introduces new challenges, particularly with regard to **reachability**. Unlike fully connected networks, a DAG does not guarantee connections between all node pairs. Due to the directed nature of the edges and the hierarchical constraints, certain nodes cannot communicate with each other, either directly or indirectly. This constraint complicates operator placement because it limits the possible assignments of computational tasks to nodes as well as the available communication paths for data exchange [9].

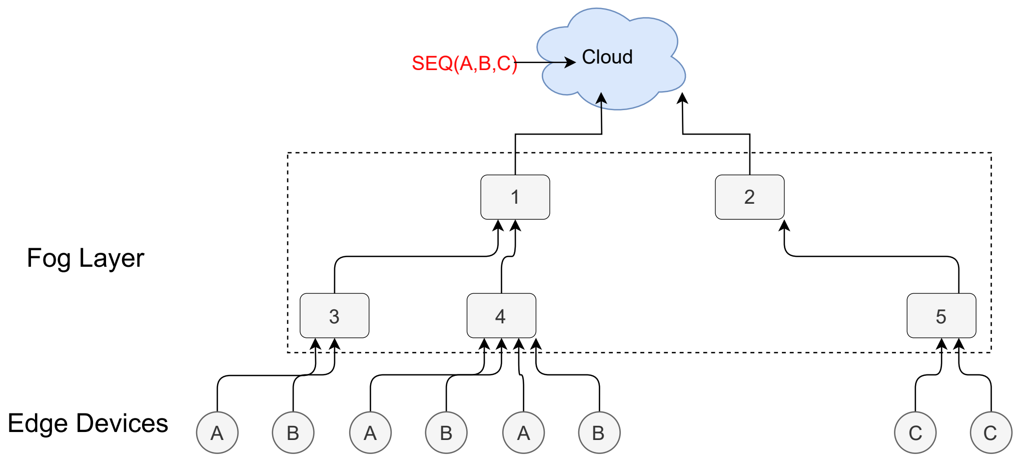


Figure 9: Example fog- cloud graph with 15 nodes

An example graph showing a basic fog cloud topology with 15 nodes is shown in Figure 9. This graph organizes the nodes into multiple levels, with directed edges indicating permissible communication paths. The lack of bidirectional connectivity and the absence of paths between certain nodes illustrate the challenges of ensuring efficient operator placement and data forwarding in this constrained topology.

A significant challenge arises when evaluating queries in this structure. For instance, consider a simple query of the form $AND(A, B, C)$, where events A, B, and C are generated within the sub-trees of nodes 1 and 2. Due to the hierarchical constraints, the only viable node for evaluating the query may be located in the cloud since the lower-level nodes lack a direct communication path between them. This situation effectively negates any potential performance improvements gained by offloading computations to fog nodes, resulting in an evaluation approach similar to a centralized cloud-based model.

Furthermore, the hierarchical nature of fog-cloud environments inherently limits the feasibility of multi-node approaches. In these network structures, multiple nodes cannot always perform query evaluation simultaneously because they cannot access all the generated events across the hierarchy. The constrained communication paths in a hierarchical network mean that events generated in different sub-trees are not accessible to multiple nodes without traversing higher levels in the hierarchy. This limitation makes multi-node methods impractical, as certain nodes would lack the necessary data to perform accurate and comprehensive query evaluations.

Therefore, we focus on single-node approaches, where a designated node evaluates the entire query. This approach aligns with the constraints of fog-cloud hierarchies, where data often funnels to a central point for processing. By strategically placing operators, we aim to maximize the use of available computational resources in the fog layer, thereby reducing the reliance on the cloud and optimizing the overall system performance.

With the given graph structure, the current system offers no advantages over centralized evaluation methods, highlighting the need for adaptations to current state-of-the-art solutions.

Prior work, such as *Network-wide Complex Event Processing over Geographically Distributed Data Sources* [10], has demonstrated potential transmission savings by combining operator placement with push-pull communication in distributed complex event processing. Our approach explores the integration of operator placement and push-pull mechanisms to address the inherent challenges of hierarchical fog-cloud environments.

4 In-Network Evaluation for Nebula Stream

In this chapter, we present our approach for combining the operator placement strategy by INEv [3] and the approach for predicate-based push-pull communication by Purtzel et al. [27] in a hierarchical fog-cloud environment. We aim to optimize event processing and communication efficiency by combining these two methodologies within the constraints and characteristics of fog-cloud networks.

We begin in Section 4.1, with an outline of the core assumptions and limitations in INEv and PrePP [3, 27] that hinder direct application in a fog-cloud topology. In Section 4.2 , we then detail the modifications and adaptations we implement to ensure these solutions are suited to the hierarchical nature of fog-cloud environments. Our codebase is publicly available as a fork of the original INEv repository and is maintained on GitHub[1].

4.1 State-of-the-Art Review: Assumptions and Limitations

We outline why modifications were necessary to adapt state-of-the-art approaches for operator placement and push-pull communication in fog-cloud environments. We discuss the underlying assumptions of each approach that do not align with the hierarchical and resource-constrained nature of fog-cloud networks.

4.1.1 In-Network-Evaluation

INEv was developed for a fog-only topology, leading to several assumptions that limit its suitability for fog-cloud environments [3]. Key assumptions include:

1. **Network Topology**: INEv assumes an undirected graph in a mesh-like structure. This structure lacks the hierarchical relationships that we have in fog-cloud settings.

[1]`https://github.com/PMeran101/INEv`

2. **Node Homogeneity**: INEv does not account for node heterogeneity. In a fog-cloud network, nodes can have varying computational and memory capacities based on their position in the hierarchy.

3. **Absence of Central Node**: INEv approach assumes a decentralized model without a central node, which does not hold true in fog-cloud settings, where the cloud orchestrates and should receive all relevant data.

These assumptions require adjustments to make their algorithms work in a fog-cloud architecture, with hierarchical data flows, heterogeneity, and a central-like cloud node.

4.1.2 Predicate-Based Push-Pull Communication for Distributed CEP

PrePP was also designed with mesh topology, assuming a flat, non-hierarchical network [27]. This design includes the following assumptions that restrict its direct application to fog-cloud environments:

1. **Network Topology**: PrePP assumes a flat non-hierarchical network topology. In which data does not need to traverse several levels to reach a certain node.

2. **Node Homogeneity**: Like INEv, PrePP assumes all nodes have equal event processing and communication capabilities. Resource allocation varies across layers in a fog-cloud network, with fog nodes often resource-constrained compared to cloud nodes.

These assumptions necessitate modifications to allow PrePP to operate efficiently within a fog-cloud environment with diverse node capacities, hierarchical data flows, and bandwidth constraints.

4.2 Proposed Adaptations and Improvements

This section provides an overview of the technical modifications implemented to adapt existing methodologies for effective operation within a hierarchical fog-cloud environment. Building on the limitations identified in the previous section, we detail the specific changes to address these challenges, including network topology, node heterogeneity, and pathfinding modifications. These adaptations are essential for enhancing communication efficiency, optimizing resource allocation, and constructing a sound simulation environment that accurately reflects fog-cloud networks' hierarchical and resource-constrained nature.

4.2.1 Node Enhancement

Encapsulating the heterogeneity of nodes in a fog-cloud network is crucial for creating a realistic simulation environment. To this end, we designed a *Node* class that models the attributes and relationships between nodes across different hierarchical levels within the network.

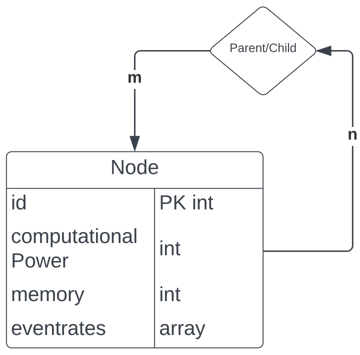


Figure 10: ER-diagram of Node class

Figure 10 illustrates our *Node* class structure as an entity-relationship diagram. The Node class possesses attributes including id, computational power, memory, and eventrates. Each attribute is defined as follows:

- **ID**: Automatically assigned at node creation, incrementally enumerating each node.
- **Computational Power and Memory**: Represent virtual capacities as integer values that vary based on the node's position in the hierarchy. Lower-layer (fog) nodes are assigned smaller computational power and memory values, with values increasing at each layer up to the cloud layer, which is assigned virtually unlimited capacity.
- **Event Rates**: An array of integers representing the types and rates of primitive events generated by the node. Source nodes generate events at specific rates, while non-source nodes have an event rate array populated with zeros.

The *Node* class incorporates a flexible parent-child relationship model, where each node instance can be assigned multiple parent and child connections. This configuration enables the realistic simulation of hierarchical, multi-connected relationships typical in fog-cloud networks.

In summary, this *Node* class structure is a robust foundation for representing the heterogeneous nature of fog-cloud environments, accommodating varying capacities and network roles across hierarchical layers.

4.2.2 Modeling Fog-Cloud Network Structure

One of the primary adaptations in our approach is the transition from an undirected graph, which is used in the original INEv implementation [3], to a **directed acyclic graph (DAG)** structure. Fog-cloud environments naturally have hierarchical structures with directed communication flows from edge devices (fog nodes) toward centralized cloud servers. By implementing a DAG, we align the network model with the directional data flow common in fog-cloud systems.

In our model, nodes interact only through predefined communication links via directed edges, representing the restricted connectivity typical in fog-cloud infrastructures. This approach ensures that data flows follow hierarchical pathways from fog to cloud, creating a structured data flow that prevents cycles and reinforces the network's layered nature. However, if beneficial, we allow pull requests to function bidirectionally to facilitate necessary data retrieval.

We use the **DiGraph** class from the **networkx** library to simulate this hierarchical structure to build a DAG-like topology. An example of such a topology is shown in Figure 9, which illustrates the organized arrangement of nodes in our simulated environment.

Another crucial parameter we integrated into this model is the **graph density**, which allows us to create more edges within our graph. We adjust the graph density by varying the maximum number of parent nodes that each child node can connect to. In a sparse configuration, each child node has only one parent node, which limits communication paths and maintains a strictly hierarchical flow. As the maximum number of parent connections increases, the network becomes denser, creating additional communication paths. These can improve the efficiency of data transmission by providing multiple routing options. This density variation allows us to examine how different levels of connectivity affect our transmission rate and other metrics.

4.2.3 Modifications to Path Finding in Fog-Cloud Architectures

The shift to a directed acyclic graph (DAG) topology introduced specific challenges in pathfinding. While the `networkx` library provides a range of algorithms suited to undirected graphs, including approximations for the **Steiner Tree Problem**—which is crucial for identifying minimal-cost sub-networks connecting specified nodes—these tools are not directly applicable in directed graph contexts. The Steiner Tree Problem is essential for minimizing communication costs within a network by finding the shortest interconnecting paths among a set of nodes [31]. However, at the time of writing, `networkx` lacks a built-in approximation for solving this problem in directed graphs.

To overcome this limitation, we implemented a custom algorithm (Algorithm 1) for computing shortest paths and identifying common ancestors within the DAG structure. This algorithm allows us to find optimal paths between nodes and pinpoint aggregate nodes that can efficiently consolidate data from multiple sources. For example, in Figure 9, nodes 3 and 5 cannot directly communicate due to the directional nature of the DAG. Thus, the only feasible node capable of aggregating events from both is node 0, the designated cloud server. This constraint highlights the importance of accurately representing directed communication flows and hierarchical dependencies in fog-cloud networks.

Aspect	**Modification/Adaptation**
Node	Created a *Node* class to represent heterogenous environment. Supports multiple parent-child connections to represent hierarchical fog-cloud structure.
Topology	Replaced undirected graph with a **DAG** to model hierarchical, directed data flow. Adjusted **graph density** by setting maximum parent nodes per child for varying connectivity.
Pathfinding	Developed a custom algorithm for shortest paths and common ancestor detection, addressing DAG pathfinding needs not met by `networkx`'s directed algorithms.

Table 2: Summary of modifications and adaptations

Table 2 summarizes our modifications and adaptions to the INEv [3] repository to resemble fog-cloud environments.

4.3 Critical Key Parameters

Our simulation environment enables dynamic adjustments to key parameters, allowing simulations to be customized to specific requirements and research objectives. We now elaborate on the impact of each parameter:

- **Network Size**: Refers to the total number of nodes in the network. Larger networks simulate more complex and realistic environments. To ensure a hierarchical topology, we calculate the number of levels in the network based on its size. This is achieved using the base 2 logarithm of the network size as an amount of levels. This ensures our hierarchical network topology.

- **Node-Event-Ratio**: Defines the probability with which each source node generates each primitive event type. For example, if there are three event types and the node-event ratio is set to 0.5, each time a source node is created, it has a 50% chance of generating each of the three possible events independently. This parameter controls the diversity of events generated by source nodes, allowing for a probabilistic selection of event types at each source node. Non-source nodes are unaffected by this ratio as they do not generate events.

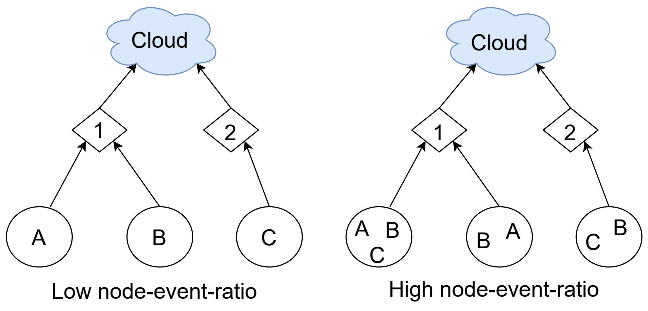


Figure 11: Exemplary network graph with increasing node-event-ratio

In Figure 11, we outline how a low node-event-ratio will result in source nodes generating fewer events, while a high node-event-ratio will result in source nodes generating more of the available event types.

- **Event Skew**: This parameter controls the skewness of the Zipf distribution used to generate event rates for primitive events. A higher skew results in greater disparities between the occurrence rates of different events, meaning some events will be much more frequent than others.

Event type	**Event skew**	
	0.1	**2**
A	1000	1000
B	1000	310
C	1000	31

Table 3: Impact of event skew on event rates.

Table 3 illustrates the event rates generated using a Zipf distribution. A lower event skew produces event rates that are nearly uniform, while higher event skew values amplify disparities, resulting in increasingly uneven eventrates.

- **Number of Event Types**: Specifies how many distinct primitive events are used in the simulation. For example, with three event types, the primitive events are labeled as A, B, and C.
- **Maximum Number of Parents**: Sets the upper limit for how many parent nodes a node can have. For instance, a maximum of two parents means each node can potentially have up to two parents. However, the exact number will vary to maintain randomness in the network generation process. Increasing this parameter results in an increased graph density.

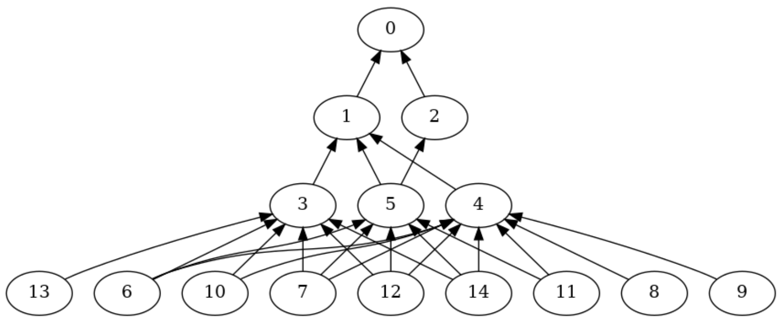


Figure 12: Examplary network graph with increased density

Figure 12 visualizes a network size of 15 and a max parents configuration of 3 and how increasing the max parents parameter increases graph density. For reference, a graph with a minimal density was displayed in Figure 9.

- **Query Length**: Sets the upper limit for a query complexity, such as count of primitive operators in one query.

Query	Low Complexity	High Complexity
Query 1	$\mathbf{SEQ}(C, M, L)$	$\mathbf{SEQ}(M, C, A, \mathbf{AND}(D, N, H))$
Query 2	$\mathbf{AND}(D, E, A)$	$\mathbf{AND}(F, \mathbf{SEQ}(G, \mathbf{AND}(M, B, L, C)))$

Table 4: Comparison of different query lengths (3 and 12).

In Table 4, we visualize the difference in query complexity we can achieve by increasing the query length parameter.

- **Query Count**: The number of queries generated to be evaluated in the environment.

In conclusion, we provide different parameters to conduct experiments for a broad series of simulations in different network settings. We summarize the parameters and their effects in the following table.

Parameter	**Effect**
Network size	Amount of nodes in the network
Node-event-ratio	Ratio for the amount of events source nodes generate
Event skew	Dictates event rate disparities
Number of event types	Number of available event types in the network
Maximum number of parents	Changes density of our network topology
Query length	Affects query complexity
Query count	Query workload, amount of queries to be executed

Table 5: Simulation parameters

4.4 Demonstrating Query Splitting with Push-Pull Communication

This section demonstrates the capabilities of INES in query splitting, operator placement, and the effective utilization of the push-pull mechanism. Our solution leverages the splitting mechanism introduced by INEv [3], ensuring optimal query distribution across the network while minimizing data transmission.

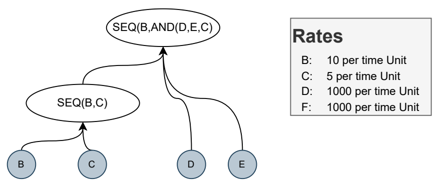


Figure 13: Operator tree

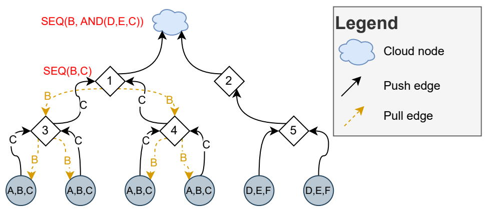


Figure 14: Network placement

Figure 13 illustrates the query splitting mechanism applied to the example query *SEQ(B, AND(D, E, C))*. The query is split into a subquery, *SEQ(B, C)*, which can be independently evaluated. This mechanism enables partial query evaluation closer to the data sources.

Figure 14 illustrates how operators are placed within the network to enhance efficiency. Node 1 handles the subquery *SEQ(B, C)*, performing preliminary evaluations that generate partial matches. This strategy reduces data transmission by filtering out irrelevant events early, leading to a more streamlined and efficient system. In node 1, we applied the push-pull communication protocol: events of type C, which occur at a lower rate, are pushed, while events of type B are pulled upon arrival of type C. The cloud completes the final query evaluation, leveraging the reduced dataset to achieve both computational and communication efficiency.

This example highlights the strength of our approach, combining query splitting, placement, and push-pull to enable scalable and effective operator placement in a distributed network environment.

4.5 Research Hypotheses to be Tested

The modifications and adaptations outlined in Table 2 are expected to influence the effects of the evaluated parameters. Based on these changes, we propose the following hypotheses, which will be addressed in detail in Section 5:

- Increasing the graph density is anticipated to reduce transmission costs for both operator placement and the push-pull mechanism.
- Push-pull communication leads to an increased latency.
- Variations in event skew are expected to reveal identifiable trends influencing system performance.

5 Analytical Evaluation

Our solution allows us to conduct several experiments to evaluate the implementation of operator placement and push-pull communication and their combination in fog-cloud environments. In this Chapter, we conduct a series of experiments and evaluate their results. Section 5.1 describes our experimental setup, and in Section 5.2, we conduct several experiments.

5.1 Experimental Setup

Data and Workload

We use purely artificial data on an abstract level. We only work with events labeled A, B, C, etc., which do not correspond to real-world events. The workload consists of various queries composed of combinations of these events, with query sizes and complexity systematically varied to assess the system's performance across different configurations.

Evaluation Metrics

The primary evaluation metrics in this study are as follows:

- **Transmission Ratio:** The transmission ratio is measured relative to a centralized, cloud-based approach. It quantifies the amount of data transmitted within the network compared to the baseline, offering insights into the efficiency of the proposed solution in reducing data transmission across the fog-cloud hierarchy.
- **Latency:** Latency is represented virtually, where each hop across an edge in the network graph increments the total latency by 1. This simplified model provides a clear, relative measure of latency within our simulated environment.
- **Computation Time:** Computation time is measured as the time required for the algorithm to find the optimal solution for a given network configuration, reflecting the computational demands of our approach.

- **Base Configuration:** Refers to the configuration of the system with only operator placement enabled, serving as a baseline to assess the impact of push-pull mechanisms.
- **Hybrid Configuration:** Refers to the configuration of the system with both operator placement and push-pull mechanisms enabled, demonstrating the combined effect of both features on the evaluation metrics.

While our primary focus is on the transmission ratio, we dedicate a separate experiment to analyzing latency and computation time to provide a comprehensive view of the solution's performance. The inclusion of Base and Hybrid configurations allows for a detailed comparison of the individual and combined contributions of the mechanisms to the observed results.

Hardware and Software Settings

Our experiments were conducted on a university-hosted cluster equipped with 60 GB of RAM and accessible via SSH. The cluster runs a Linux-based operating system.

Simulation Environment

Our simulation environment was configured to optimize reproducibility and scalability. Using Python 3.10, we leveraged the following libraries:

- `NumPy` for efficient data generation and manipulation,
- `NetworkX` for network topology creation.

The simulations were orchestrated via shell scripts.

5.2 Experiments

Our methodology is based on a simulation-based study. The experiments are carried out in a controlled environment where the settings can be systematically varied as described in Table 5. We simulate networks with different configurations and evaluate the performance in terms of data transmission efficiency.

We evaluate INES, which represents a fog-cloud environment with the new features presented in Table 2.

Each experiment is structured into three parts:

1. **Experiment Setup** - describing the unique characteristics, such as network size, number of event types, and query size
2. **Observation** - presenting the plots of the results and explaining the observed trends.
3. **Conclusion** - deriving insights from the results and discussing their implications.

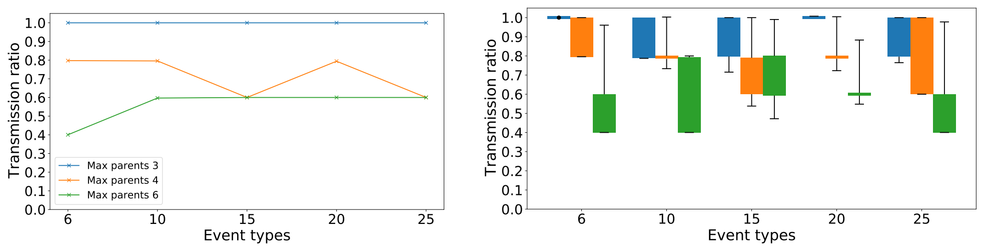
Our graphs include average line graphs, bar graphs, and whisker graphs, which show 90% quantiles along with the min and max values, to convey statistical relevance.

5.2.1 Increasing Graph Density on Base Approach

Setup

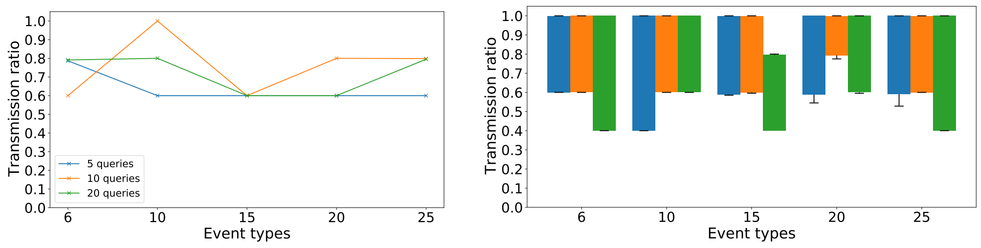
In our first experiment, we investigate how the density of the network graph affects the overall performance of operator placement, specifically the efficiency of data transmission. We start with a sparse configuration, where nodes are minimally connected, and gradually increase the number of edges to simulate a denser network. This stepwise approach allows us to analyze how improved connectivity affects the transmission ratio, an important key performance indicator.

The network size is fixed at 50 nodes, and we explore different configurations to simulate various network scenarios. The number of event types varies between 5 and 25, while the query counts are set to 5, 10, and 20. By adjusting these parameters, we systematically investigate how increased graph density affects data transmission efficiency under different workloads and event distributions. Our goal is to better understand the relationship between connectivity and transmission ratio.



(a) (b)

Figure 15: Transmission ratios by max parents across event types.



(a) (b)

Figure 16: Transmission ratios by workload size across event types.

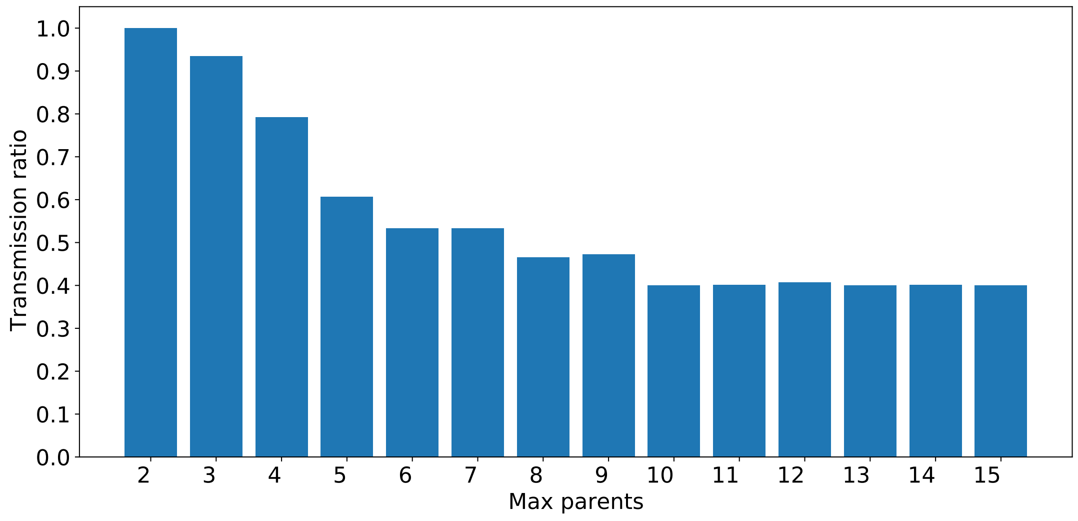
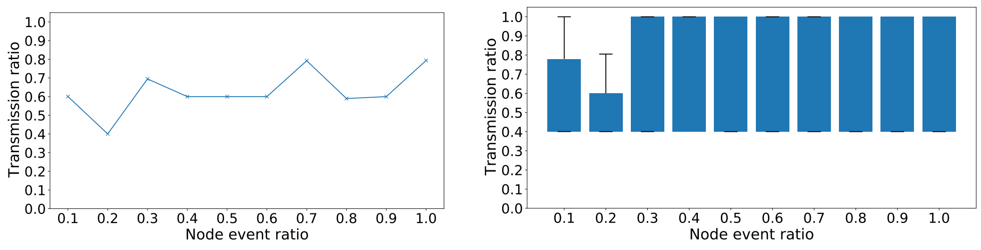


Figure 17: Average transmission ratio for max parents configurations.



(a) (b)

Figure 18: Transmission ratio by node-event-ratio configurations.

Observation

In Figure 15a, the transmission ratio is compared across different configurations of max parents. For a network size of 50, a max parents setting of 3 shows no noticeable difference in transmissions compared to a centralized cloud architecture. However, as the graph density increases (i.e., max parents settings of 4 and 6), we observe a reduction in transmission ratio. Our Base solution offers a reduction in transmission ratio of up to 60%. For a max parents configuration of 4, we observe a minimum improvement of transmission ratio of 20%. This suggests that increasing graph density can improve transmission efficiency. Figure 15b further supports this observation, as the underlying data demonstrates that higher max parents values consistently result in a lower transmission ratio, highlighting the positive impact of greater node connectivity on transmission efficiency.

Figure 16 examines the effect of varying query sizes on the transmission ratio for a fixed max parents value of 4. This max parents setting was chosen because it was the lowest graph density, offering significant improvements to the cloud-based approach, as we wanted to study the effects of other parameters in the network. The line graph in Figure 16a illustrates that the transmission ratio increases significantly as query workload grows. However, the statistical analysis shown in Figure 16b reveals that the 90% quantile transmission ratio remains nearly constant across all query sizes. This finding suggests that, while query workload affects the average transmission ratio, the distribution of transmission efficiency within the network remains relatively stable across different query sizes.

Figure 17 presents the average transmission ratio for various query sizes and max parents values. The results indicate a gradual decrease in transmission ratio as the max parents parameter increases, demonstrating that higher graph density reduces transmission overhead. However, beyond a max parents setting of 10, no further improvements in the transmission ratio are observed.

Lastly, Figure 18 presents the transmission ratios for experiments conducted with different node-event ratios. One can observe that there is no correlation between the node-event ratio and the transmission ratio.

Conclusion

The experiment underlines that increasing the graph density consistently reduces the transmission ratio, thus confirming the hypothesis that stronger connectivity improves the efficiency of data flow in fog-cloud networks. Higher graph density enables more direct communication paths between nodes and significantly reduces data transfer needs. This is particularly important in distributed environments such as fog-cloud networks because minimizing transmissions leads to less congested networks.

A key takeaway is the 'sweet spot' at a maximum parent of 10 for a network size of 50 nodes. Beyond this point, further increases in graph density do not yield additional improvement in transmission efficiency, revealing a ceiling on the benefits of increased connectivity. This provides a practical guideline for network design: increasing graph density up to this threshold optimizes efficiency without introducing unnecessary complexity.

In smaller networks, the node-event ratio does not impact the transmission rate and can therefore be neglected in design decisions. These results provide the basis for further investigations in more realistic environments. The experiment 5.2.4 builds on these findings by examining larger networks to determine whether the sweet spot remains consistent and how the trends change with increasing network size.

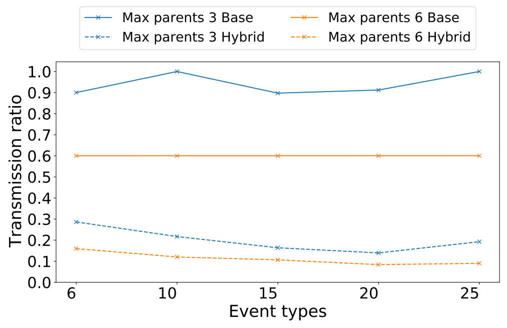
Finally, the experiment confirms that increasing graph density is a practical and effective strategy to optimize the transmission efficiency in CEP systems. These findings pave the way for more efficient and scalable designs in fog-cloud architectures.

5.2.2 Impact of Varying Graph Density on Hybrid Configuration

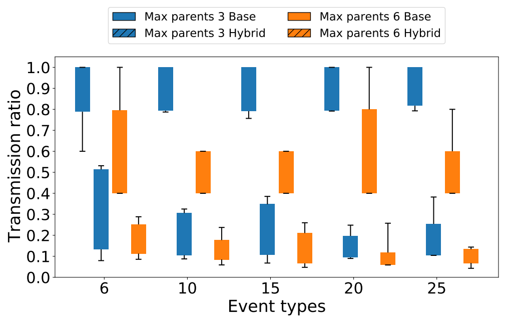
Setup

The second experiment investigates the effect of incorporating a push-pull communication strategy on data transmission within networks with different settings. In push-pull, INES utilizes the modifications made to PrePP [27] to compute which events shall be pushed and which are pulled. This mechanism aims to optimize communication by selectively transmitting data only when specific conditions, such as matching query requirements, are met.

The experiments are conducted on networks with a fixed size of 50 nodes with varying key parameters: the number of event types (ranging from 5 to 25), query workloads (5, 10, and 20), event skew factors, and different levels of graph density (as defined in Experiment 1). The push-pull communication model is introduced to the existing configurations to assess how it interacts with network density to influence the transmission ratio. By comparing the results with and without the push-pull mechanism, the experiment aims to evaluate the potential of this strategy for reducing unnecessary data transmission and improving network performance.

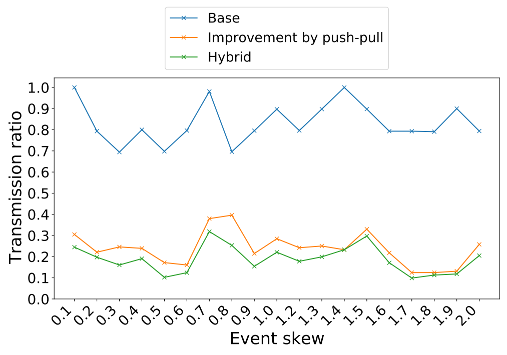


(a)

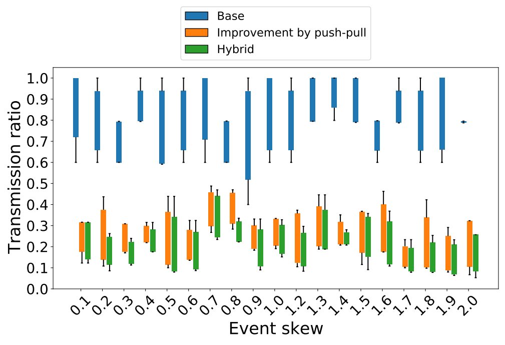


(b)

Figure 19: Base and Hybrid vs. event types and densities.



(a)



(b)

Figure 20: Base and Hybrid configurations across event skew.

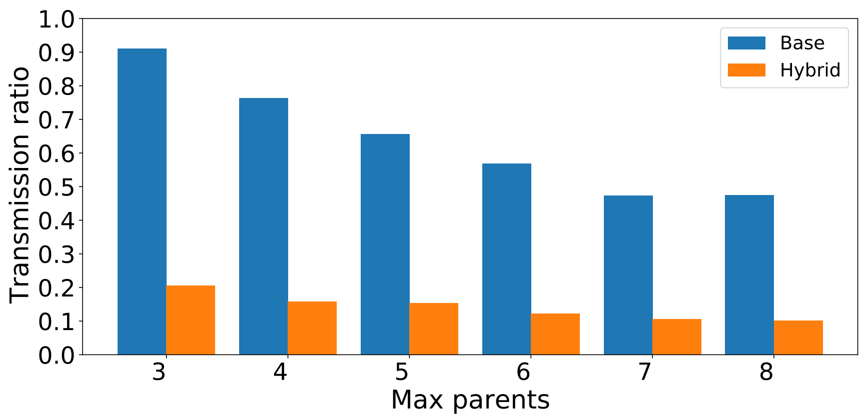
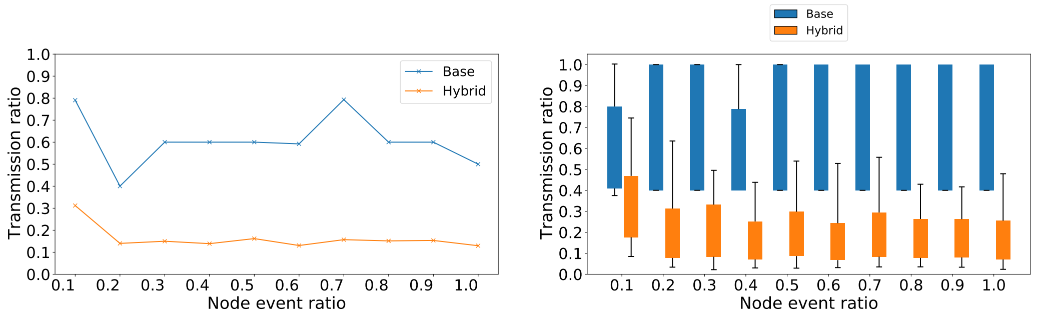


Figure 21: Base and Hybrid across max parents.



(a) (b)

Figure 22: Base and Hybrid for different node-event-ratio.

Observation

Comparative analyses of the transmission ratio with and without the push-pull mechanism reveal substantial improvements in data transmission efficiency. Incorporating the push-pull strategies proposed by PrePP [27] significantly reduces the transmission ratio compared to employing the operator placement approach alone. As depicted in Figure 19a, the combination of push-pull and operator placement reduces the transmission ratio by up to 90%. Figure 19b further demonstrates the statistical significance of this reduction, indicating that the integration of push-pull consistently lowers the transmission ratio across different scenarios. An examination of network density effects shows that increasing the number of connections (e.g., the network's density) does not produce the same level of improvement in transmission ratio as observed with operator placement alone. While an increase in network density leads to a slight reduction in the transmission ratio, the effect is not as substantial as with operator placement. Specifically, increasing the maximum number of parents from 3 to 8 reduces the transmission ratio by approximately 10%, whereas operator placement achieves a more significant reduction of around 40%.

Figure 20 illustrates the impact of varying event skew parameters on the transmission ratio for both the operator placement alone and the combined approach. The results indicate that the event skew parameter does not significantly influence the transmission ratios when the push-pull mechanism is employed.

Additionally, Figure 21 compares the reduction in transmission ratio between the sole operator placement solution and the combined solution. The findings show a substantial decrease in transmission Ratio with the use of push-pull strategies. However, increasing the network density—represented by the maximum parent parameter—does not yield proportionally significant improvements. Specifically, setting the maximum parent parameter to 3 results in an 80% reduction in transmission ratio, while increasing it to 8 leads to an 85% reduction, an additional improvement of only 5%.

Similar to Experiment 5.2.1 the node-event-ratio does not show to have an effect on the system performance in smaller network settings, see Figure 22. There is no evidence that the transmission ratio will change accordingly if the node-event ratio is lower or higher.

Conclusion

Integrating push-pull mechanisms with operator placement strategies significantly reduces data transmissions in our fog-cloud environments. The experimental results indicate that this combined approach can reduce the transmission ratio by up to 90%, significantly outperforming the use of solely using operator placement. Statistical validation, as shown in Figure 19b, confirms this consistency and reliability of these improvements across different configurations.

The increase in density does not affect the transmission ratio as significantly as the operator placement. Specifically, increasing the maximum parent parameter from 3 to 8 yields only an additional 5% reduction, suggesting limited returns beyond a certain network density. Furthermore, the negligible impact of varying event skew parameters on the transmission ratio, as illustrated in Figure 20, highlights the robustness of the push-pull mechanism against different eventrates.

In conclusion, combining push-pull communication and operator placement is a highly effective strategy for reducing transmissions in a fog-cloud network. They offer a significant efficiency gain in most network configurations.

5.2.3 Effect of INES on Latency and Computation Time

Setup

While the incorporation of the push-pull mechanism into our operator placement solution significantly reduces the transmission ratio, we hypothesize that these improvements may introduce additional costs in terms of latency or computation time. The push-pull strategies could potentially increase coordination overhead and processing delays due to the more complex communication patterns they introduce. To investigate this potential trade-off, this experiment aims to evaluate the impact of integrating push-pull mechanisms on both latency and execution time metrics within the network.

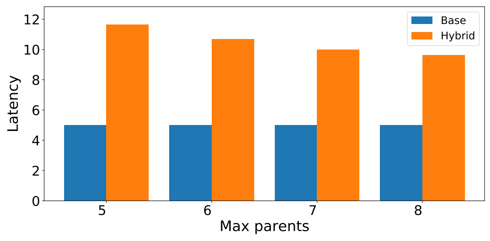


Figure 23: Latency on increasing density

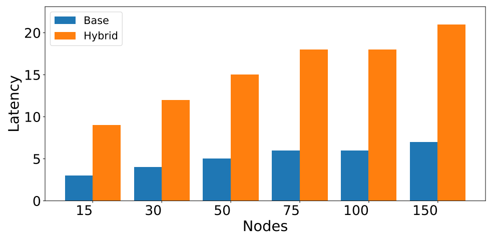


Figure 24: Latency on increasing network sizes with 1 max parent.

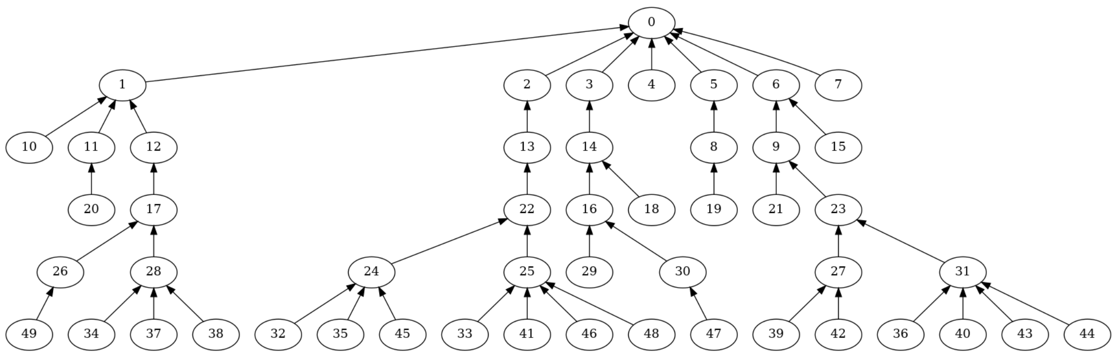


Figure 25: Network of 50 nodes with a max parent of 1

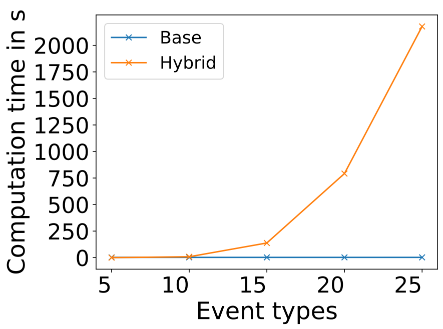
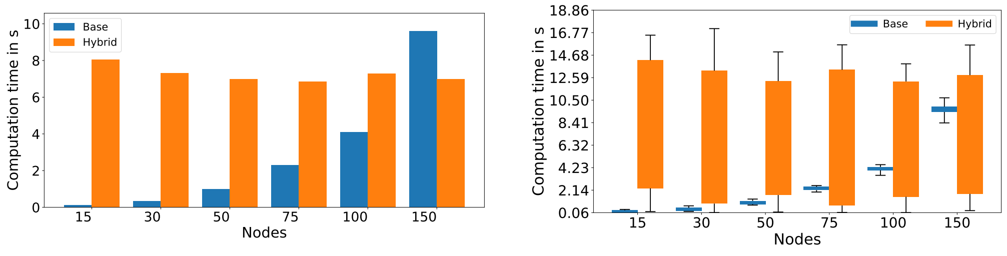


Figure 26: Computation time for Base and Hybrid with 50 nodes and 1 max parent.



(a) (b)

Figure 27: Computation time vs. network size for Base and Hybrid.

Observation

Figure 23 demonstrates that increasing graph density leads to a reduction in latency for the push-pull communication approach, while latency remains constant for the operator placement method. This behavior can be explained using Figure 25. In the operator placement strategy, events are pushed directly to a designated node, and ultimately all data is forwarded to the cloud. Since the communication path remains consistent, the latency does not vary with graph density. In contrast, the push-pull approach introduces additional latency due to its reliance on a selected node for event aggregation. In this method, an event is pushed to the selected node, which then pulls the remaining events before sending the aggregated data to the cloud. When the selected node is the cloud itself—common in low-density graphs where the maximum parent variable is set to one—this results in latency up to three times higher than that of operator placement. This increase arises because events must traverse multiple network levels to reach the topmost cloud node.

However, as the graph density increases, the selected node for push-pull communication is positioned lower in the network hierarchy, reducing the overall latency by shortening the communication paths. Additionally, as shown in Figure 24, increasing the network size while maintaining low density consistently results in latency that is three times higher for push-pull communication compared to operator placement. This is due to the sparse connectivity of low-density networks, which limits the choice of aggregation nodes and often forces reliance on the cloud as the central processing unit.

Our experiments show that configurations with a high number of event types significantly increase computation time for the push-pull mechanism, as illustrated in Figure 26. The complexity of managing multiple event types during execution drives this exponential growth in computation time.

Figure 27 highlights that increasing the number of nodes in the network minimally impacts the computation time required to calculate the optimal plan for the push-pull model. However, adding more nodes drastically increases the time needed to place operators. This indicates that the push-pull model efficiently handles computation planning as the network grows, but operator placement becomes increasingly more comples in larger topologies.

Conclusion

As demonstrated in Experiment 5.2.2, integrating the push-pull mechanism into our model significantly reduces the number of transmissions in the network. However, this comes at the cost of increased latency due to the pulling of events. This effect can be observed in Figure 23 and 24.

We observe that increasing graph density helps reduce latency. This is because computational tasks are executed lower in the hierarchy, resulting in fewer network hops. Additionally, Figure 26 shows an exponential increase in computation time for the push-pull plan creation as the number of event types grows. Networks with fewer event types exhibit correspondingly lower computation times.

Interestingly, the number of event types does not significantly affect the computation time needed to determine the optimal operator placement. However, as illustrated in Figure 27a, increasing the network size does not impact the push-pull implementation's computation time but instead increases the time required to compute the optimal operator placement.

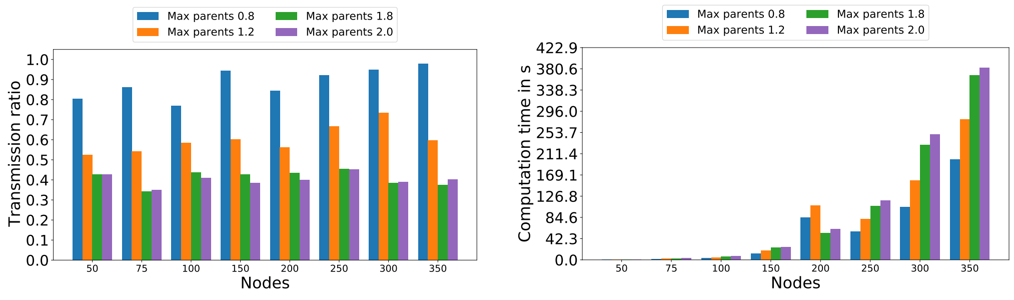
5.2.4 Scaling INES to Moderate Sizes

Setup

This experiment investigates the scaling capabilities of INES by focusing on varying network sizes and densities. Given the substantial computational demands associated with larger networks—where processing times for networks of 1,000 nodes can exceed three hours—we limit the scope of this experiment to a maximum network size of 350 nodes. This constraint ensures a practical and efficient evaluation while preserving sufficient variability to capture meaningful performance trends.

The experiment evaluates the impact of network size and density on three key metrics: transmission ratio, system latency, and computational time. To enable a unified comparison across these parameters, the maximum number of parents is defined as a relative value based on a factor of the network's levels. This approach ensures a more adaptive and scalable configuration, accommodating variations in network size and density while maintaining consistency in evaluation.

By carefully examining these metrics under the defined constraints and using this adaptive method for setting parent node limits, the experiment provides valuable insights into the performance, trade-offs, and limitations of INES under moderate scaling conditions. This controlled setup enables a thorough investigation of the system's behavior while maintaining computational feasibility.



(a) (b)

Figure 28: Base transmission ratio and computation time

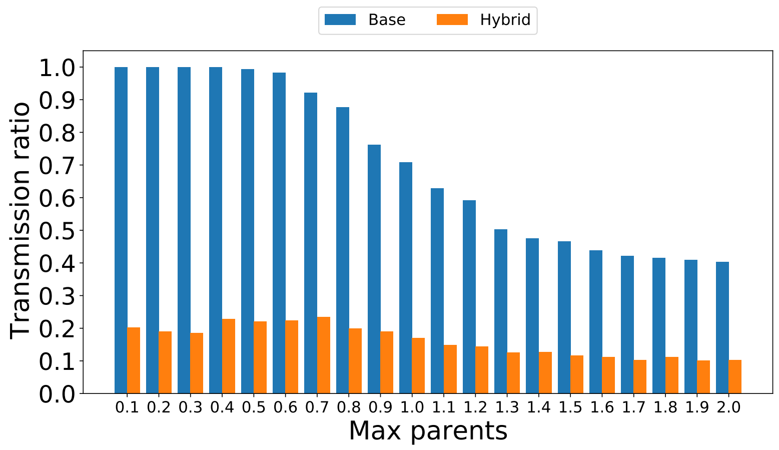
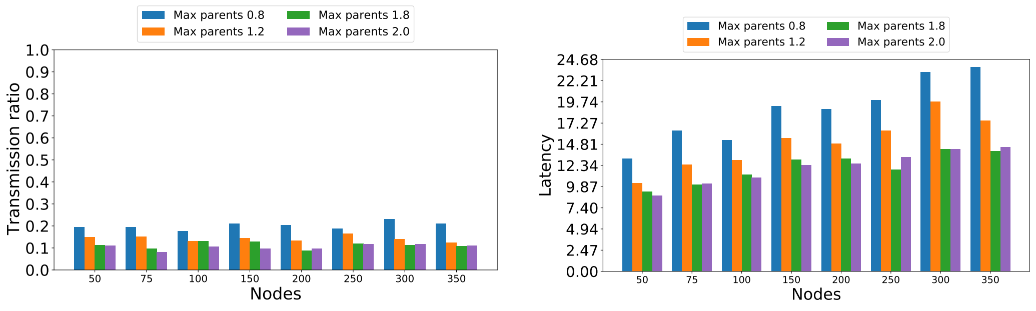


Figure 29: Average transmission ratio for different max parents



(a) (b)

Figure 30: Hybrid transmission ratio and latency

Observation

We observe in Figure 28a that, irrespective of the network size, the Base approach consistently achieves up to a 70% reduction in transmission ratio. Additionally, the results demonstrate that for various network sizes, the transmission ratio does not improve significantly once the maximum parent factor reaches 1.8.

Figure 28b reveals that increasing network size exponentially increases the computation time for the Base approach, and this effect is further intensified with higher network density. The outlier for low density at the network size of 200 is caused by a server crash.

Furthermore, Figure 29 confirms that a maximum parent factor of 1.8, irrespective of the network sizes, yields optimal performance without increasing network complexity unnecessarily. It is important to note that the graph displays the data using a complete average over all network sizes, ensuring that this conclusion is not biased toward any specific network size. This observation applies to both the Base and Hybrid approaches, with the Base approach benefiting more from increased complexity than the Hybrid approach.

Finally, Figure 30 highlights a strong correlation between network density and the Hybrid approach. While the Hybrid approach provides limited benefits in reducing transmission ratio, it significantly reduces latency, demonstrating its primary advantage.

Conclusion

Based on the findings of this experiment, we conclude that the Base approach achieves significant reductions in transmission ratio, with improvements of up to 70%. This reduction remains robust across varying network sizes, demonstrating the approach's reliability. The results further indicate that increasing the maximum parent factor beyond 1.8 offers diminishing returns in transmission ratio, establishing 1.8 as the optimal threshold for this parameter.

While the Base approach benefits more from increased network complexity compared to the Hybrid approach, it incurs exponentially higher computation time as network size and density increase. This trade-off emphasizes the need to balance computational overhead with transmission efficiency when scaling the network. Figure 28b highlights the necessity of refining our algorithm to better manage scaling in both node count and density.

The Hybrid approach achieves up to a 90% reduction in transmission ratio, demonstrating its effectiveness in minimizing network overhead. However, it exhibits high latency when operating with a low maximum parent factor. As shown in Figure 29, increasing density in the Hybrid approach yields an additional 10% reduction in transmissions. More notably, latency decreases by almost 30% when density increases from a factor of 0.8 times the level count to 1.8 times the level count.

In conclusion, we demonstrate the effectiveness of our solution in handling scaling, reducing transmissions, and minimizing latency. However, improving the solution's computation time remains a critical area for future work.

5.3 Key Takeaways

Our experiments analyzed how INES performs across varying network settings. We conclude that INES significantly reduces transmission costs compared to a purely central approach in most cases. We demonstrated that INES can achieve up to 90% transmission savings, although with some latency trade-offs.

The effects of key parameters on our solution are summarized in Table 6:

Parameter	**Impact on INES**
Network Size	- Consistently achieves transmission savings. - Computation for operator placement increases significantly.
Node-Event Ratio	- No correlation observed.
Event Skew	- No significant impact.
Number of Event Types	- No impact on transmission ratio. - Greatly increases computation time for push-pull.
Max Parents	- Most crucial parameter for transmission ratio. - Higher values improve savings and reduce push-pull latency.
Query Length	- Increases complexity. - Raises overall computation time.
Query Count	- Increases complexity and computation time.

Table 6: Key takeaways from experiments

These findings underline the importance of tailoring specific parameters, such as 'max parents', to optimize transmission savings and reduce latency, while managing the trade-offs in computation time for larger network configurations.

6 Related Work

Research for CEP strategies is growing, especially for distributed networks. The potential of combining operator placement and push-pull was studied in several works [7, 10]. In this section, we place our work into the scientific context. Both comet and network-wide, complex event processing strategies were studied in a distributed network environment. Whereas our work focuses on combining these strategies in the fog-cloud setting.

Network-wide complex event processing over geographically distributed data sources

Building on the foundational concepts of operator placement and push-pull, *Network-wide Complex Event Processing over Geographically Distributed Data Sources by Flouris et al.* advances these ideas by developing a framework optimized specifically for CEP over large-scale, geographically dispersed networks. Their approach distinguishes itself by tightly integrating in-network processing with a push-pull mechanism, forming a comprehensive, bi-criteria optimization model that minimizes network bandwidth consumption while meeting latency requirements dictated by SLA constraints.

The novelty of this work lies in its systematic integration of service-oriented objectives with CEP operations, a departure from traditional models that often consider only communication cost or network latency in isolation. By structuring the operator placement problem as a constrained optimization that balances both network load and QoS, the study adapts dynamically to environments with stringent service agreements. This is further facilitated by a combination of exhaustive algorithms for exact solutions and scalable heuristics, which allow for near-optimal placement configurations under computational constraints. This dual-algorithmic approach provides theoretical and practical pathways for efficiently handling event streams, adapting seamlessly across different network structures and performance demands.

Furthermore, this work's practical impact is underscored by its deployment within the FERARI multi-cloud platform, where the authors validate their methods using real data and topologies. These experiments confirm that Flouris et al.'s integrated approach significantly outperforms conventional centralized and in-network models, achieving substantial reductions in both data transmission and event detection latency.

In the broader context of related work, this study extends the core concepts of operator placement and push-pull into a holistic, service-oriented framework. While previous methods have focused individually on aspects of CEP optimization, this work effectively combines these elements, offering a structured solution for complex, distributed environments. It represents a forward step in CEP research, providing a scalable and robust model for managing real-time data in geographically expansive systems [10].

Comet: Decentralized Complex Event Detection in Mobile Delay Tolerant Networks

The Comet framework [7] introduces a novel decentralized approach for complex event detection (CED) in mobile delay-tolerant networks (DTNs), addressing key limitations of centralized methods. Traditional centralized CED approaches face significant challenges in DTNs due to inherent characteristics such as long delays, frequent disconnections, and high communication costs. These limitations render centralized frameworks unsuitable for scenarios where scalability and efficiency are paramount.

Comet's design distributes the CED process across multiple DTN nodes, enabling the detection of complex events through a hierarchical structure called a CED tree. This approach leverages intermediate nodes for event aggregation, reducing communication overhead and improving efficiency. Using Dijkstra's shortest path algorithm, the framework's unique *h-function* combines cost and delay metrics to construct cost-effective and delay-sensitive CED trees. Junction nodes in these trees host sub-CED processes, allowing partial event detection closer to the data sources.

A critical innovation of Comet is its two-phase push-pull conversion algorithm. In the first phase, proactive event pushes are replaced with single-target pulls wherever feasible, minimizing detection costs without exceeding specified delay tolerances. The second phase reduces costs by converting remaining push operations into multi-target pulls, provided delay constraints are met. This heuristic approach ensures that event processing is efficient and adaptable to the dynamic characteristics of DTNs.

Experimental evaluations demonstrate Comet's superior performance over traditional centralized methods, including all-push and heuristic-based algorithms. Compared to all-push strategies, Comet achieves up to 89% cost reduction and outperforms centralized optimal approaches by approximately 56%. The framework's ability to effectively balance cost and delay underscores its potential for real-world DTN applications.

7 Conclusion

As the number of devices connected to the internet continues to grow, real-time data evaluation faces increasing challenges in ensuring timely and efficient processing. The fog-cloud topology, which combines the strengths of fog and cloud computing, offers a viable solution to address these challenges. However, for Complex Event Processing (CEP) to function effectively in such environments, existing algorithms must be adapted to meet the specific requirements of fog-cloud architectures and utilize their additional ressources.

Methodology and Contributions. This work introduces INES (In-Network Evaluation for Nebula Stream), an extension of INEv [3], tailored for fog-cloud environments. To achieve this, we identify and overcome several limitations of the original INEv framework, designed for mesh-like fog-only networks. We implemented a DAG-like topology to represent data flow, added bidirectional pull requests, and integrated PrePP's [27] algorithm to enable push-pull communication. These contributions form the foundation of our open-source framework, which is available for further research and development.

Findings and Results. Our experiments demonstrate that INES, which integrates both operator placement and push-pull strategies, achieves a reduction in data transmissions of up to 90% compared to a centralized cloud approach. This significant improvement is made possible by the combined effect of these two strategies. Additionally, we show that incorporating push-pull mechanisms on top of operator placement yields an additional 80% reduction in transmissions compared to using operator placement alone. Although these strategies increase latency and computational overhead, they enable significant pre-processing within the fog layer, reducing the load on the network. Our findings highlight that network density plays a critical role in achieving transmission efficiency, as outlined in Table 6 (see Section 5.3). In scenarios with constrained communication paths, performance aligns more closely with centralized architectures, underscoring the importance of adequate connectivity. We observe that graph density significantly impacts operator placement but has a lesser effect on push-pull communication. While push-pull greatly reduces transmissions, it increases latency, reflecting a trade-off between transmission efficiency and response time. However, INES consistently reduces transmission costs as network size increases, consistently outperforming centralized approaches.

Limitations and Lessons Learned. Despite its successes, INES has limitations. The current implementation relies on brute-force methods, making it computationally expensive for large-scale networks. This made a simulation for realistic network sizes for 1,000 nodes or more unrealistic. Additionally, latency variations between edges are not yet modeled, which limits the framework's applicability in latency-sensitive environments. These challenges highlight the need for more scalable algorithms and realistic latency modeling in future work. A key lesson from this research is that effective fog-cloud infrastructures require flexible and dense communication paths to ensure efficiency, especially for distributed processing tasks.

Future Work. Future research can explore several directions to build upon this work. One promising area involves evaluating the impact of additional edges in the network, which could further reduce transmission costs and optimize data flow. Developing more efficient algorithms to replace brute-force methods is another critical task, enabling scalability for larger networks. For instance, the current approach, which requires approximately three hours to process a network with 1000 nodes and a maximum parent of 8, highlights the necessity for optimization. Finally, incorporating custom latency parameters for specific edges, such as those between the fog layer and the cloud, could create more realistic simulations and provide deeper insights into latency-sensitive scenarios.

Summary. In summary, this work demonstrates the effectiveness of combining operator placement and push-pull in one strategy to optimize data transmission in fog-cloud environments. While there are limitations, INES provides a strong foundation for future research and highlights the potential of adaptive and scalable fog-cloud architectures to address the challenges of real-time data processing in distributed systems.

Bibliography

[1] Ahammad, I.: Fog computing complete review: Concepts, trends, architectures, technologies, simulators, security issues, applications, and open research fields. SN Comput. Sci. 4(6), 765 (2023), https://doi.org/10.1007/s42979-023-02235-9

[2] Akdere, M., Çetintemel, U., Tatbul, N.: Plan-based complex event detection across distributed sources. Proc. VLDB Endow. 1(1), 66–77 (2008), http://www.vldb.org/pvldb/vol1/1453869.pdf

[3] Akili, S., Purtzel, S., Weidlich, M.: Inev: In-network evaluation for event stream processing. Proc. ACM Manag. Data 1(1), 101:1–101:26 (2023), https://doi.org/10.1145/3588955

[4] Akili, S., Weidlich, M.: Muse graphs for flexible distribution of event stream processing in networks. In: Li, G., Li, Z., Idreos, S., Srivastava, D. (eds.) SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021. pp. 10–22. ACM (2021), https://doi.org/10.1145/3448016.3457318

[5] Alli, A.A., Alam, M.M.: The fog cloud of things: A survey on concepts, architecture, standards, tools, and applications. Internet Things 9, 100177 (2020), https://doi.org/10.1016/j.iot.2020.100177

[6] Bierzynski, K., Escobar, A., Eberl, M.: Cloud, fog and edge: Cooperation for the future? In: Second International Conference on Fog and Mobile Edge Computing, FMEC 2017, Valencia, Spain, May 8-11, 2017. pp. 62–67. IEEE (2017), https://doi.org/10.1109/FMEC.2017.7946409

[7] Chen, J., Ramaswamy, L., Lowenthal, D.K., Kalyanaraman, S.: Comet: Decentralized complex event detection in mobile delay tolerant networks. In: Aberer, K., Joshi, A., Mukherjea, S., Chakraborty, D., Lu, H., Venkatasubramanian, N., Kanhere, S.S. (eds.) 13th IEEE International Conference on Mobile Data Management, MDM 2012, Bengaluru, India, July 23-26, 2012. pp. 131–136. IEEE Computer Society (2012), https://doi.org/10.1109/MDM.2012.18

[8] 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

[9] 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

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

[11] Gong, C., Liu, J., Zhang, Q., Chen, H., Gong, Z.: The characteristics of cloud computing. In: Lee, W., Yuan, X. (eds.) 39th International Conference on Parallel Processing, ICPP Workshops 2010, San Diego, California, USA, 13-16 September 2010. pp. 275–279. IEEE Computer Society (2010), https://doi.org/10.1109/ICPPW.2010.45

[12] Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future Gener. Comput. Syst. 29(7), 1645–1660 (2013), https://doi.org/10.1016/j.future.2013.01.010

[13] He, Z., Peng, L.: Evaluation of fog topologies in fog planning for iot task scheduling. In: Hung, C., Cerný, T., Shin, D., Bechini, A. (eds.) SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing, online event, [Brno, Czech Republic], March 30 - April 3, 2020. pp. 2177–2180. ACM (2020), https://doi.org/10.1145/3341105.3374027

[14] He, Z., Zhang, Y., Tak, B., Peng, L.: Green fog planning for optimal internet-of-thing task scheduling. IEEE Access 8, 1224–1234 (2020), https://doi.org/10.1109/ACCESS.2019.2961952

[15] Herbst, N.R., Kounev, S., Reussner, R.H.: Elasticity in cloud computing: What it is, and what it is not. In: Kephart, J.O., Pu, C., Zhu, X. (eds.) 10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013. pp. 23–27. USENIX Association (2013), https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst

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

[17] Lakshmanan, G.T., Li, Y., Strom, R.E.: Placement strategies for internet-scale data stream systems. IEEE Internet Comput. 12(6), 50–60 (2008), `https://doi.org/10.1109/MIC.2008.129`

[18] Lange, M., Koschel, A., Astrova, I.: Dealing with data streams: Complex event processing vs. data stream mining. In: Gervasi, O., Murgante, B., Misra, S., Garau, C., Blecic, I., Taniar, D., Apduhan, B.O., Rocha, A.M.A.C., Tarantino, E., Torre, C.M., Karaca, Y. (eds.) Computational Science and Its Applications - ICCSA 2020 - 20th International Conference, Cagliari, Italy, July 1-4, 2020, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 12252, pp. 3–14. Springer (2020), `https://doi.org/10.1007/978-3-030-58811-3_1`

[19] Lepping, A.P., Pham, H.M., Mons, L., Rueb, B., Grulich, P.M., Chaudhary, A., Zeuch, S., Markl, V.: Showcasing data management challenges for future iot applications with nebulastream. Proc. VLDB Endow. 16(12), 3930–3933 (2023), `https://www.vldb.org/pvldb/vol16/p3930-lepping.pdf`

[20] Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., Zhao, W.: A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. IEEE Internet Things J. 4(5), 1125–1142 (2017), `https://doi.org/10.1109/JIOT.2017.2683200`

[21] Markl, V.: Nebulastream - data stream processing in massively distributed, heterogeneous, volatile environments. In: Proceedings of the 18th ACM International Conference on Distributed and Event-based Systems, DEBS 2024, Villeurbanne, France, June 24-28, 2024. pp. 1–3. ACM (2024), `https://doi.org/10.1145/3629104.3672505`

[22] Mell, P., Grance, T.: The nist definition of cloud computing (2011-09-28 2011)

[23] Muratev, H.: Survey of state-of-the-art distribution strategies for complex event processing (2024)

[24] Nardelli, M., Cardellini, V., Grassi, V., Presti, F.L.: Efficient operator placement for distributed data stream processing applications. IEEE Trans. Parallel Distributed Syst. 30(8), 1753–1767 (2019), `https://doi.org/10.1109/TPDS.2019.2896115`

[25] O'Keeffe, D., Salonidis, T., Pietzuch, P.R.: Frontier: Resilient edge processing for the internet of things. Proc. VLDB Endow. 11(10), 1178–1191 (2018), `http://www.vldb.org/pvldb/vol11/p1178-okeeffe.pdf`

[26] 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

[27] Purtzel, S., Akili, S., Weidlich, M.: Predicate-based push-pull communication for distributed CEP. In: Zhou, Y., Chrysanthis, P.K., Gulisano, V., Zacharatou, E.T. (eds.) 16th ACM International Conference on Distributed and Event-based Systems, DEBS 2022, Copenhagen, Denmark, June 27 - 30, 2022. pp. 31–42. ACM (2022), https://doi.org/10.1145/3524860.3539640

[28] Riaz, F., Ali, K.M.: Applications of graph theory in computer science. In: Al-Dabass, D., Suwarno, Yunus, J., Saad, I., Giriantari, D., Abraham, A. (eds.) Third International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN 2011, Bali, Indonesia, 26-28 July, 2011. pp. 142–145. IEEE Computer Society (2011), https://doi.org/10.1109/CICSyN.2011.40

[29] Starks, F., Goebel, V., Kristiansen, S., Plagemann, T.: Mobile distributed complex event processing - ubi sumus? quo vadimus? In: Skourletopoulos, G., Mastorakis, G., Mavromoustakis, C.X., Dobre, C., Pallis, E. (eds.) Mobile Big Data, A Roadmap from Models to Technologies, Lecture Notes on Data Engineering and Communications Technologies, vol. 10, pp. 147–180. Springer (2018), https://doi.org/10.1007/978-3-319-67925-9_7

[30] Tawsif, K., Hossen, J., Raja, J.E., Jesmeen, M.Z.H., Arif, E.M.H.: A review on complex event processing systems for big data. In: 2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP). pp. 1–6 (2018)

[31] Voß, S.: The steiner tree problem with hop constraints. Ann. Oper. Res. 86, 321–345 (1999), https://doi.org/10.1023/A%3A1018967121276

[32] 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. www.cidrdb.org (2020), http://cidrdb.org/cidr2020/papers/p7-zeuch-cidr20.pdf

[33] Zhang, W., Chen, X., Jiang, J.: A multi-objective optimization method of initial virtual machine fault-tolerant placement for star topological data centers of cloud systems. Tsinghua Science and Technology 26(1), 95–111 (2021)

Appendix A. Further Details on the Solution Approach

Algorithm 1: Find Shortest Path or Ancestor

```
Input: $G$: Graph
$me$: Node from which the path starts
$j$: Target node
Output: List of nodes representing the path between $me$ and $j$
Function find_shortest_path_or_ancestor($G$, $me$, $j$):
    // Try to find a direct shortest path
    if direct path from me to j exists then
        return routingAlgo[me][j] // Return the direct shortest
            path
    else
        // Find the shortest combined path via an ancestor or
            fallback to node 0
        if common ancestors between me and j exist then
            $min\_distance \leftarrow \infty$
            $shortest\_combined\_path \leftarrow \emptyset$
            foreach ancestor in common ancestors do
                $path\_me\_to\_ancestor \leftarrow \text{routingAlgo}[me][ancestor]$
                $path\_j\_to\_ancestor \leftarrow \text{routingAlgo}[j][ancestor]$
                $combined\_distance \leftarrow \text{length}(path\_me\_to\_ancestor) +$
                    $\text{length}(path\_j\_to\_ancestor) - 2$ // Subtract 2 to
                    avoid double counting the ancestor
                if $combined\_distance < min\_distance$ then
                    $min\_distance \leftarrow combined\_distance$
                    $shortest\_combined\_path \leftarrow \text{path\_me\_to\_ancestor} +$
                        $\text{reverse(path\_j\_to\_ancestor)[1:]}$ // Combine paths
                        without duplicating the ancestor
                end
            end
            return $shortest\_combined\_path$
        else
            // Fallback to cloud node 0 if no common ancestor
                exists
            $path\_me\_to\_cloud \leftarrow \text{routingAlgo}[me][0]$
            $path\_j\_to\_cloud \leftarrow \text{routingAlgo}[j][0]$
            return $path\_me\_to\_cloud + reverse(path\_j\_to\_cloud)[1:]$
        end
    end
```