# INEv: In-Network Evaluation for Event Stream Processing

Anonymous Author(s)

## ABSTRACT

Complex event processing (CEP) detects situations of interest by evaluating queries over event streams. Once CEP is used in networked applications, the distribution of query evaluation among the event sources enables performance optimization. Instead of collecting all events at one location for query evaluation, sub-queries can be placed at network nodes to reduce the data transmission overhead. Yet, existing techniques either place such sub-queries at exactly one node in the network, which neglects the benefits of truly distributed evaluation, or are agnostic to the network structure, which neglects transmission costs incurred by the absence of direct network links.

To overcome the above limitations, we propose INEV graphs for in-network evaluation of CEP queries with rich semantics, including Kleene closure and negation. Our idea is to introduce fine-granular routing of *partial* results of sub-queries as an additional degree of freedom in query evaluation: We exploit events already disseminated in the network as part of one sub-query, when evaluating another one. We show how to instantiate INEv graphs by splitting a query workload into sub-queries, placing them at network nodes, and forwarding of their results to other nodes. Also, we characterize INEv graphs that guarantee correct and complete query evaluation, and discuss their construction based on a cost model that unifies transmission and processing latency. Our experimental results indicate that INEv graphs can reduce transmission costs for distributed CEP by up to eight orders of magnitude compared to baseline strategies.

## CCS CONCEPTS

• **Information systems** → **Data streams**.

## KEYWORDS

event stream processing, operator placement, in-network evaluation

**ACM Reference Format:**

Anonymous Author(s). 2022. INEv: In-Network Evaluation for Event Stream Processing. In *Proceedings of (SIGMOD '23).* ACM, New York, NY, USA, 16 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

Complex event processing (CEP) comprises models and methods to evaluate queries over streams of event data [25]. It enables the detection of *situations of interest* in applications that span, for instance, electrical grids [22], finance [45], and urban transportation [9]. In essence, a CEP query matches a pattern of events, characterized in terms of the types of events, their ordering, and a time window.



*SIGMOD '23, June 18–23, 2023, Seattle*



ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Adopting CEP in networked applications that integrate multiple event sources (e.g., smart meters or payment terminals) [43] induces a design space related to the location of query evaluation:

A *centralized* model evaluates queries at a single network node over a unified stream [25] and, hence, it suffers from limited scalability: It requires sending *all* events over the network, although most of them are not required to construct the matches of a query.

A *distributed* model exploits event processing capabilities of network nodes [23]. Data transmission is reduced by assigning (partial) queries to network nodes. The construction of a plan for distributed query evaluation is challenging, though, due to the large search space induced by the following design choices:

D1 How to *split* a query workload for distribution: One may consider queries in isolation [44], take into account sharing opportunities between queries [36], split each query according to its operator hierarchy [23], or consider query rewritings [4, 41].

D2 How to *place* (sub-)queries in the network: For a query, one may consider single-node [21, 34] or multi-node placements [4, 5].

D3 How to *forward* query matches between network nodes to assemble the matches: Here, one may assume direct links between all nodes [5] or consider the network topology [23, 34].

The design choices shall be taken such that query evaluation is correct and complete, stays within a certain latency bound, and minimizes the data transmission between nodes. Yet, already the optimization of the placement of operators is an NP-hard problem [11].

There is a notable research gap, since exploration of the above design space has been limited. Specifically, Flouris et al. [23] incorporate solely simple splits of a query and single-node placements. These limitations are addressed by MuSE graphs [4] that feature query rewritings and multi-node placements, but neglect the network topology and assume that all nodes can communicate directly.

In this paper, we address this gap with a novel model for in-network evaluation of CEP workloads. It is based on the following insight: To exploit the optimization potential of query rewritings and multi-node placements under arbitrary network topologies, fine-granular forwarding of *partial* matches of sub-queries is required. Compared to existing work on distributed CEP, we hence add an additional degree of freedom in query evaluation to leverage the dissemination structure of events in the network:

- We propose INEv graphs as a model for in-network evaluation of a CEP workload. INEv graphs include an explicit selection of partial matches for forwarding between nodes.
- We formally show correctness and completeness of the model, and provide a cost model to reason on its efficiency.
- We present an algorithm to construct an efficient INEv graph, which approximates an intractable optimal solution and exploits sharing opportunities between multiple queries. We also present strategies for adapting INEv graphs to react to network changes.

We evaluate INEv graphs in simulation experiments and with an implementation that is based on a framework for distributed computing. Our results indicate that INEv graphs reduce transmission costs by up to eight orders of magnitude compared to baseline strategies.

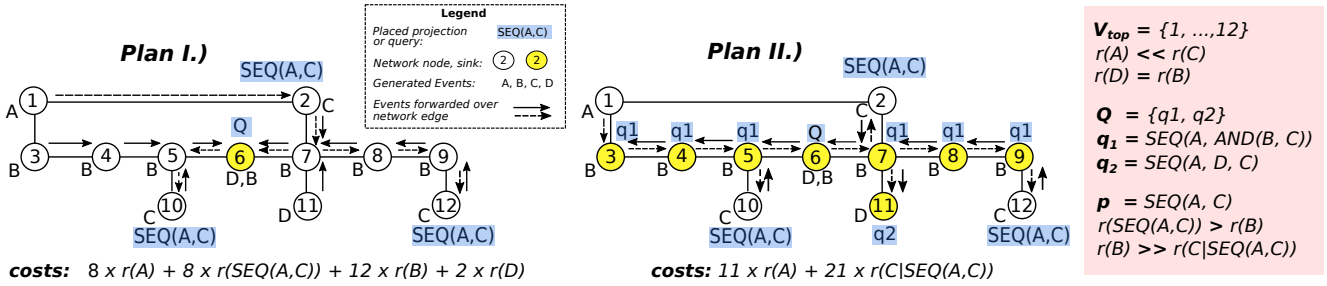



**Figure 1: Motivating example for in-network evaluation of CEP queries.**

In the remainder, we first provide a motivating example (§2), give preliminaries of distributed CEP (§3), and formalize the addressed problem (§4). We then introduce INEv graphs for in-network evaluation of CEP queries and study their properties (§5), before turning to their construction (§6). Finally, we discuss experimental results (§7), review related work (§8), and conclude the paper (§9).

## 2 MOTIVATING EXAMPLE

Consider a networked application in the context of smart grids, i.e., demand response (DR) management [26]. Here, consumers shall react to requests by a utility to reduce their energy consumption for a certain time period to avoid costly handling of peak demands. CEP may help to monitor these responses and enables immediate reaction in case of non-conforming behaviour [37].

Figure 1 illustrates a network of event sources, where a controller (node 1) generates events about DR requests (events of type $A$). The other nodes denote consumers, classified as households (nodes 3-9) that report their base consumption ($B$). Some households are linked to further devices (nodes 2, 10-12), such as charging points for vehicles, that schedule energy consumption for short ($C$) and long ($D$) durations. The rates of event generation differ. DR requests ($A$) occurr with a much lower rate than consumption events ($B$-$D$).

For monitoring responses to DR requests, a workload $Q$ of CEP queries, such as $q_1$ and $q_2$ in Figure 1, is evaluated. The queries detect patterns of events representing requests ($A$) and subsequent events about current ($B$) and scheduled ($C$, $D$) consumption.

**Centralized model.** For centralized evaluation, all relevant events are gathered at one node. Aiming at minimal transmission costs, the node that generates the largest portion of the relevant events and has the shortest distance to other nodes qualifies for query evaluation. In our example, node 6 is suitable, as it generates events of two types ($B$, $D$) and is the most central node in the network. The transmission cost of centralized query evaluation at node 6 is given by the rates of the event types and the number of hops that the respective events need to be sent over the network, i.e., $3 \cdot r(A) + 12 \cdot r(B) + 8 \cdot r(C) + 2 \cdot r(D)$.

**Distributed models.** Query evaluation as sketched in plans I and II realizes distributed CEP. The plans incorporate query rewriting when *splitting* the queries for distribution, i.e., queries $q_1$ and $q_2$ are evaluated based on the sub-query $SEQ(A, C)$. Using $SEQ(A, C)$, which is not part of the operator hierarchy of $q_1$, for the evaluation of both queries also leverages sharing opportunities for $q_1$ and $q_2$. The plans adopt a multi-node *placement*, i.e., $SEQ(A, C)$ is placed at all source nodes of $C$ events (2, 10, 12). As these events occur with a high rate, sending the rare $A$ events to sources of $C$ reduces transmission costs compared to a single-node placement of $SEQ(A, C)$.

Yet, a simple mechanisms to *forward* query results yields superfluous data transmission. In plan I, which forwards $A$ events to each node evaluating sub-query $SEQ(A, C)$ via the shortest path (as illustrated with dashed arrows), the results of sub-query $SEQ(A, C)$ and all $B$ and $D$ events are forwarded to node 6, which evaluates the queries $q_1$ and $q_2$. This plan has a transmission cost of $8 \cdot r(A) + 8 \cdot r(SEQ(A, C)) + 12 \cdot r(B) + 2 \cdot r(D)$, thereby improving over the centralized evaluation plan (no transmission of high-rate $C$ events). However, the plan ignores the potential for optimization that is induced by the $A$ events already disseminated in the network.

In plan II, events of type $A$, needed at nodes 2, 10, and 12 to evaluate $SEQ(A, C)$, are sent along a routing tree that contains all nodes generating $B$ and $D$ events (again, illustrated with dashed arrows). As such, each of these nodes receives all $A$ events. Therefore, instead of forwarding the matches of $SEQ(A, C)$, it suffices to forward *partial* results, i.e., all $C$ events that are part of the matches. The rate of these $C$ events is smaller than the rate with which matches of $SEQ(A, C)$ materialize, since a single event may, in general, be part of a large number of matches. With the rate of the respective $C$ events being smaller than those of $B$ and $D$ events, multi-node placements of queries $q_1$ and $q_2$ at nodes generating either $B$ or $D$ events become viable. With these multi-node placements, the transmission cost of plan II is $11 \cdot r(A) + 21 \cdot r(C|SEQ(A, C))$, which, compared to plan I, saves the rates of sending $B$ and $D$ events.

## 3 PRELIMINARIES

### 3.1 System Model

An *event* is an occurrence of a phenomena, for which the semantics is given by an *event type* [7, 25]. The latter defines a schema, a set of attributes. We write $\mathcal{E} = \{\epsilon_1, \ldots, \epsilon_n\}$ for the set of event types.

An *event network* $\Gamma = (G_{top}, f, r)$ is defined by a topology graph $G_{top} = (V_{top}, E_{top})$, a function $f : \mathcal{E} \rightarrow 2^{V_{top}}$ that assigns event types to (source) nodes, and a function $r : \mathcal{E} \rightarrow \mathbb{R}$ that assigns an average occurrence rate to an event type. We consider event networks, in which event sources also have computational capabilities and assume homogeneity in their ability to take part in the query evaluation.

The topology graph $G_{top}$ is a connected, undirected graph. Any exchange of events between two nodes $n, m \in V_{top}$ has to happen along a path in $E_{top}$. With $r(\epsilon)$ as the *local rate* for an event type $\epsilon \in \mathcal{E}$, we refer to $R(\epsilon) = |f(\epsilon)| \cdot r(\epsilon)$ as the *global rate* for the network-wide occurrences of an event type. Thus, we assume that the number of occurrences per time unit is roughly the same for events of the same type; and later explore in experiments whether our approach is robust against this assumption being not satisfied.

EXAMPLE 1. *Figure 1 illustrates an event network with event types* $\mathcal{E} = \{A, B, C, D\}$ *and nodes* $\{1, \ldots, 12\}$*, where, for instance,* $f(D) = \{6, 11\}$*. Also, we already discussed that* $r(C) \gg r(A)$*. For node* 12 *to send events to node* 7*, the events need to be forwarded along the path* $\langle (12, 9), (9, 8), (8, 7) \rangle$.

To define semantics of queries over events generated in a network, a mechanism to order the events from different nodes needs to be in place, e.g., based on timestamps (breaking ties deterministically). Such a mechanism shall provide *some* order, but is not required to resolve issues, such as clock skew. Also, out-of-order arrival of events shall be handled by common techniques, such as watermarking.

## 3.2 Query Language

We adopt a common query model for CEP [8, 25], defined as follows.

**Syntax.** A query $q = (O, \beta)$ is an ordered tree of operators, annotated with predicates and a time window. The operators $O$ are the vertices of the query tree. We distinguish composite operators $O_c$ that encode some ordering for child operators, and primitive operators $O_p$ that refer to an event type. The function $\beta : O_c \twoheadrightarrow O^k$ with $k \in \mathbb{N}$ assigns a sequence of child operators to a composite operator. A composite operator $o \in O_c$ has a semantic type, denoted by $o.sem \in \{AND, SEQ, OR, NSEQ, KL\}$.

We write $\mathcal{E}(q)$ for the set of event types referenced by primitive operators of query $q$. To simplify the notation, we assume that each event type is referenced by at most one primitive operator of a query.

**Semantics.** Evaluating a query yields a set of *matches*, each match being a sequence of events that have been generated at the network nodes. In addition to sequencing, conjunction, and disjunction, we consider the Kleene closure $KL$ of one primitive operator, and $NSEQ$ that negates an operator between two other operators, see [31]. The matches for a query are defined inductively over the operator tree:

- A primitive operator creates a match for each event of the event type referenced by the operator.
- $AND$ constructs a match for the interleaving of matches of all its children.
- $SEQ$ constructs a match for the concatenation of matches of its children in the specified order.
- $OR$ constructs a match for each match of one of its children.
- $KL$ constructs a match for each (arbitrary long) sequence of matches of its children.
- $NSEQ$ constructs a match for a sequence of matches of its first and last child, if no match of the middle child occurs in between.

The above construction is applied only for events that satisfy the predicates and the time window defined by the query. Our semantics corresponds to a *greedy* event selection strategy, also known as *unconstrained* [14] or *skip-till-any-match* [1]. It does not impose any bound on the number of times an event can participate in matches.

Given an event network $\Gamma = (G_{top}, f, r)$, the evaluation of a query $q = (O, \beta)$ is modelled by the *rate*, with which matches are generated. For a primitive operator $o \in O_p$ that references an event type $\epsilon \in \mathcal{E}$, the rate corresponds to the local rate of the type in the network and the operator selectivity $\sigma(o)$, which is induced by the predicates defined the query over the attributes of the event type, i.e., $r(o) = \sigma(o) \cdot r(\epsilon)$. For a composite operator $o \in O_c$ with $\lambda(o) = \langle o_1, \ldots, l_k \rangle$ and selectivity $\sigma(o)$, the rate is defined inductively:

**Table 1: Overview of notations for networks and queries.**

| Notation | Explanation |
|---|---|
| $\mathcal{E} = \{\epsilon_1, \ldots, \epsilon_n\}$ | Universe of event types |
| $\Gamma = (G_{top}, f, r)$ | Event network: topology graph $G_{top} = (V_{top}, E_{top})$, sources of event types $f : \mathcal{E} \to 2^{V_{top}}$, rates $r : \mathcal{E} \to \mathbb{R}$ |
| $r(\epsilon)$, $R(\epsilon)$ | Local and global rate of an event type $\epsilon$ |
| $q = p = (O, \beta)$ | Query $q$ (or projection $p$) with composite and primitive operators $O = O_c \cup O_p$ and their tree structure $\lambda : O_c \twoheadrightarrow O^k$ |
| $\mathcal{E}(q)$, $\mathcal{E}(p)$ | Event types of a query $q$ and a projection $p$ |
| $\mathcal{Q}$ | Query workload: a set of queries |
| $c = (\mathfrak{P}, \lambda)$ | Combination with projections $\mathfrak{P}$ and inputs $\lambda$ |
| $\Pi_q$, $C_q$, $\Pi_\mathcal{Q}$, $C_\mathcal{Q}$ | All possible projections and combinations of query $q$ and query workload $\mathcal{Q}$ |
| $\sigma(o)$, $\sigma(q)$, $\sigma(p)$ | Selectivity of an operator, a query, or a projection |

$$r(o) = \begin{cases} \sigma(o) \cdot k \cdot \prod_{1 \le i \le k} r(o_i) & \text{if } o.sem = AND, \\ \sigma(o) \cdot \prod_{1 \le i \le k} r(o_i) & \text{if } o.sem = SEQ, \\ \sigma(o) \cdot \sum_{1 \le i \le k} r(o_i) & \text{if } o.sem = OR, \\ \sigma(o) \cdot r(o_1) \cdot r(o_3)) & \text{if } o.sem = NSEQ, \\ \sigma(o) \cdot 2^{r(o_1)} & \text{if } o.sem = KL. \end{cases}$$

Based thereon, the rate $r(q)$ and the selectivity $\sigma(q)$ of a query $q$ are defined as the rate and selectivity of its root operator, respectively.

**Query workload.** In the remainder, we consider a query workload $\mathcal{Q}$, i.e., a set of queries that do not contain any $OR$ operator. Any query containing an $OR$ operator can be split into multiple queries, so that this does not constrain our model's expressiveness. Also, we evaluate any query $q \in \mathcal{Q}$ with the largest time window defined by any query in $\mathcal{Q}$. As such, sharing of intermediate results is enabled at the expense of result filtering based on the original window and potentially higher data transmission. We later assess the impact of the differences in time windows within a query workload empirically.

## 3.3 Query Splitting and Placement

To enable distributed CEP, a query is split into sub-queries, which are then placed at network nodes for evaluation [4].

**Query projections and combinations.** We consider a mechanism to split a query $q$ into *projections*, where a projection $p = (O, \beta)$ is a query itself that is restricted to a subset of the event types referenced in $q$, i.e., $\mathcal{E}(p) \subset \mathcal{E}(q)$. A projection also inherits the predicates of $q$ defined over $\mathcal{E}(p)$, as well as the time window of $q$. In contrast to hierarchical notions of sub-patterns [38], matches of a projection are not necessarily contiguous sub-sequences of query matches. We denote the set of all possible projections for a given query $q$ by $\Pi_q$.

Projections that jointly realize correct and complete evaluation of a query are captured by a *combination* [4]. We formalize a combination $c = (\mathfrak{P}, \lambda)$ of a query $q$ as a directed acyclic graph (DAG) having projections $\mathfrak{P} \in \Pi_q$ as its vertices. The function $\lambda : \Pi_q \to \Pi_q^k$ with $k \in \mathbb{N}, k > 1$, assigns to a projection $p$ its predecessors in $c$, which we refer to as *inputs* of $p$, as matches of these predecessors are used to construct the matches of $p$. In contrast to the operator tree of a query, a combination is formalized as a DAG as matches of one projection can be input to the generation of matches of multiple other projections. We denote the set of all combinations of a query $q$ and of all queries of a workload $\mathcal{Q}$ as $C_q$ and $C_\mathcal{Q}$, respectively. For a workload, sharing opportunities are exploited by representing a projection only once in the combination of the workload, even if it would be part of the combinations of multiple queries.

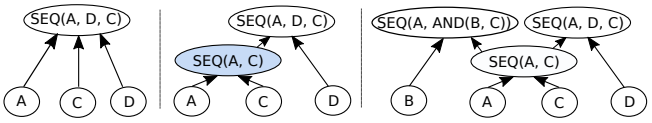



**Figure 2: Query tree of the query $q_2$ of Figure 1 (left); a combination for $q_2$ (middle) and for the query workload $Q$ (right).**

Example 2. *Figure 2 shows a combination for distributed evaluation of $q_2$ of our example; and a combination for the workload $Q$. The projection $SEQ(A, C)$ is used to evaluate both queries in $Q$.*

Below, we consider a function $split : Q \rightarrow C$ for the decomposition of a workload query into a combination for distribution.

**Placements.** To evaluate a query in a network $\Gamma = (G_{top}, f, r)$ with $G_{top} = (V_{top}, E_{top})$, based on a combination $c = (\mathfrak{P}, \lambda)$, a *placement* assigns the respective projections to network nodes. That is, a function $place : \Pi_q \rightarrow 2^{V_{top}}$ defines for each projection $p \in \mathfrak{P}$ a set of network nodes to evaluate $p$.

We distinguish between single-node and multi-node placements for a projection $p$. In a single-node placement, exactly one node $n$ evaluates $p$ and *all* events of the inputs of $p$ are gathered at node $n$. In a multi-node placement, a set of nodes collectively evaluates $p$, so that the matches of $p$ are partitioned between multiple nodes. Specifically, this partitioning is induced by one of the inputs $i \in \lambda(p)$ of $p$, referred to as the *partitioning input*, and the placement is given by the respective sources of $i$. As such, a multi-node placement avoids that events of the partitioning input are sent over the network. However, to ensure that all matches of $p$ are generated, events of all other, non-partitioning inputs of $p$ need to be forwarded to the sources of the partitioning input. From that it follows that multi-node placements are beneficial, when the rates of some event types (to be used as partitioning inputs) are significantly higher than those of others, as high-rate event types are not longer sent over the network.

Example 3. *In Figure 1, the rate of C events is high. Hence, in plan I, there is a multi-node placement for $SEQ(A, C)$ at source nodes of C, so that a sub-query matching only C events is the partitioning input. For $q_1$ and $q_2$, a single-node placement at node 6 is used, which requires sending all matches of $SEQ(A, C)$ as well as all B and D events to node 6.*

## 4 PROBLEM STATEMENT

We formalize the in-network evaluation problem for CEP queries as follows. Given an event network $\Gamma = (G_{top}, f, r)$ and query workload $Q$, this problem is grounded in three functions:

- A function *split*, see §3.3, to compute a combination for $Q$.
- A function *place*, see §3.3, to compute a placement for the projections of a given combination at the nodes of $\Gamma$.
- A function *forward* to compute a sub-graph of the topology graph $G_{top}$ for a given placement $place(p)$ of a projection $p$, which determines how to send matches of the inputs $\lambda(p)$ of $p$ to the nodes that evaluate $p$, given by $place(p)$.

In-network evaluation aims to reduce the rate with which events are sent over the network. Given an instantiation of the three functions *split*, *place* and *forward*, distributed evaluation of workload $Q$ in the network $\Gamma$ has a *transmission cost*, denoted by $\psi \in \mathbb{R}$, that captures the total rate with which nodes send events over the network.

**Table 2: Notations for in-network query evaluation.**

| Notation | Explanation |
|---|---|
| $\psi$ | Transmission cost of workload $Q$ in event network $\Gamma$ |
| $\ell$ | Latency of workload $Q$ in event network $\Gamma$ |
| $os(p)$ | Output selector: Maps a set of event types or $\equiv$ to a projection $p$ |
| $I = (V, E)$ | INEv graph with vertices $V$ and edges $E$; a vertex $v = (v_{proj}, v_{node}, v_{graph}, v_{os}) \in V$ represents a projection, network node, a forwarding graph and an output selector |

Example 4. *For plan I in Figure 1, splitting returns the combination shown in Figure 2 (right). The placement includes, e.g., $place(q_2) = \{6\}$ and $place(SEQ(A, C)) = \{2, 10, 12\}$. For the placement of sub-query $SEQ(A, C)$ at node 2, forwarding is modelled by the graph $G_{sub} = (\{1, 2\}, \{(1, 2)\})$, sending A events to node 2.*

Moreover, query evaluation induces a latency with which matches of queries are generated [48]. For distributed CEP, the latency induced by centralized evaluation of a query workload denotes a meaningful bound to consider. For an instantiation of the functions *split*, *place* and *forward*, the *latency* of the obtained plan to evaluate query workload $Q$ in the network $\Gamma$ is captured as $\ell \in \mathbb{R}$. Let $\tau \in \mathbb{R}$ be the latency of the centralized evaluation of $Q$ in $\Gamma$, i.e., no splitting of queries, all queries are placed at a single node, and all events generated in the network are sent to this node. Based thereon, we formulate the problem of in-network evaluation of CEP queries:

Problem 1. *Let $\Gamma$ be an event network and $Q$ be a query workload. The problem of in-network evaluation of $Q$ in $\Gamma$ is to instantiate functions split, place, and forward, s.t. $\psi$ is minimal and $\ell \leq \tau$.*

## 5 IN-NETWORK EVALUATION GRAPHS

To address the above problem, we propose the model of an *in-network evaluation (INEv) graph*. Given a query workload and an event network, such a graph formalizes a combination of the workload, a placement of the respective projections, and the sub-graphs used to forward the inputs of each projection. Below, we first introduce output selectors used to guide the forwarding of events (§5.1). Then, we define INEv graphs and show their correctness and completeness (§5.2). Based on models for the transmission cost (§5.3) and latency (Example 5.3), we elaborate on optimal INEv graphs (§5.4).

### 5.1 Output Selectors

When forwarding events during in-network evaluation of a query, we aim at exploiting the resulting dissemination for query evaluation. Specifically, events that are forwarded in the network are used in the evaluation of all relevant projections placed at network nodes on the path. Hence, such a node no longer needs to receive the events of all types of all inputs of the respective projections placed at it.

To realize this idea, we introduce the notion of an *output selector*. For two projections $p, p' \in \Pi_q$ of a query $q$, one being an input of the other one, $p \in \lambda(p')$, it limits the types of events that are sent from nodes hosting the projection $p$ to nodes hosting the projection $p'$ that is evaluated based on $p$. Formally, an output selector is a function $os : \Pi_q \rightarrow 2^{\mathcal{E}(q)} \cup \{\equiv\}$. It assigns to a projection $p \in \Pi_q$ a subset of event types to forward or the symbol $\equiv$, which indicates that no output selection is applied. As such, a node hosting the projection $p$

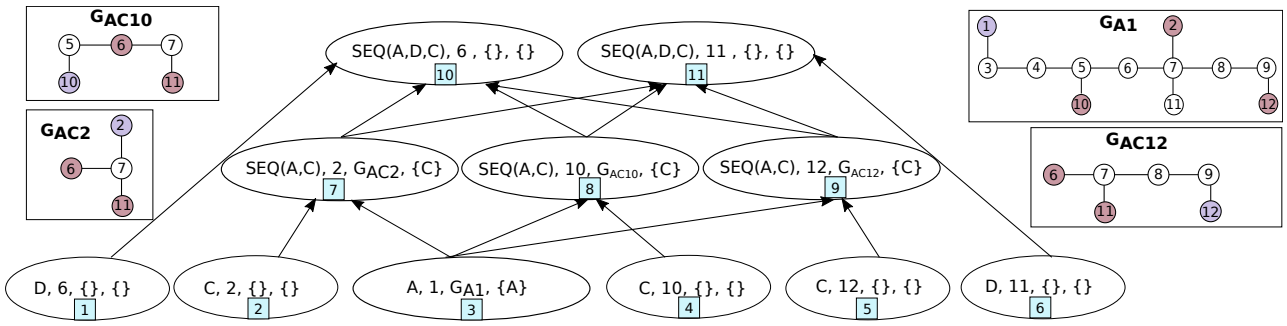



**Figure 3: INEv graph for a query workload $Q = \{q_2\}$ and the event network of Figure 1.**

either forwards the individual events of the types mentioned in the output selector, or the matches of the projection.

Example 5. *In plan II of Figure 1, A events are forwarded along all nodes generating B and D events. As such, for example, node 6 receives all A events and, thus, needs only the C events contained in matches of SEQ(A, C) to locally generate the matches of $q_1$, i.e., SEQ(A, D, C). Hence, the evaluation of SEQ(A, C) at nodes $\{2, 10, 12\}$ incorporates an output selector $os(SEQ(A, C)) = \{C\}$, thereby exploiting that the A are already present at node 6.*

## 5.2 INEv Graphs

**Definition.** An in-network query evaluation (INEv) graph captures, for a given event network $\Gamma$ and query workload $Q$, an instantiation of the three functions *split*, *place* and *forward*, see §4. As such, it defines (i) the combination used to decompose the workload $Q$; (ii) the placement for each projection of the combination; and (iii) the paths used to forward events in the network to evaluate the projections along with an output selector per projection.

Definition 1 (INEv Graph). *Let $\Gamma = (G_{top}, f, r)$ an event network with the topology graph $G_{top} = (V_{top}E_{top})$. Moreover, let $G^{sub}_{top}$ be the set of all sub-graphs of $G_{top}$. Given a query workload $Q$, an INEv graph is a directed, acyclic graph $I = (V, E)$ with vertices $V \subseteq \Pi_Q \times V_{top} \times G^{sub}_{top} \times (2^{\mathcal{E}} \cup \{\equiv\})$, and edges $E \subseteq V \times V$.*

Given an INEv graph $I = (V, E)$, we use the following notation for a vertex $v = (v_{proj}, v_{node}, v_{graph}, v_{os}) \in V$: $v_{proj}$ is the projection; $v_{node}$ is the network node; $v_{graph}$ is the sub-graph of the topology graph; and $v_{os}$ the output selector of the projection.

Exploiting existing results for query splitting and placement for distributed CEP, see §3.3, parts of the structure of an INEv graph are derived directly from the definition of a query workload and an event network. That is, an INEv graph contains a vertex for each pair of an event type referenced in a query and any node in the network that is a source of this type. Moreover, there is a vertex for each projection of the combination and the nodes of the placement of the projection. Edges in an INEv graph capture dependencies between projections. That is, for two vertices $v, v' \in V$, a directed edge $(v, v') \in E$ denotes that matches of $v_{proj}$ are used to generate matches of $v'_{proj}$.

However, INEv graphs go beyond existing models by incorporating mechanisms to guide the forwarding of events in the network. This is realized by specifying the sub-graph $v_{graph}$ and the output selector $v_{os}$ for each vertex $v$ in the INEv graph. We illustrate the intuition behind them with an example.

Example 6. *Figure 3 shows an INEv graph for the query workload $Q = \{q_2\}$ with $q_2 = SEQ(A, C, D)$ according to plan II of Figure 1. It contains a vertex for each pair of the event types A, C and D, and their sources in the graph $G_{top}$ (vertices 1-6). Moreover, it contains vertices 7-9 for the placement of projection $p_1 = SEQ(A, C)$ at nodes 2, 10, and 12; vertices 10 and 11 to place $q_2$ at nodes 6 and 11. Since matches of $q_2$ are generated using matches of $p_1$, there is an edge between the respective vertices. A vertex of a projection that is sent over the network contains a sub-graph of $G_{top}$ that shows over which edges the projection is forwarded, e.g., $G_{AC2}$ for vertex 7 defines that matches of SEQ(A, C) generated at node 2 are sent to the nodes 6, 7, and 11.*

*The last entry of each vertex defines the selected output. For projection $p_1$, only the C events in matches of $p_1$ are forwarded, as indicated by the output selectors of vertices 7-9. Also, if outputs are not sent due to them being a partitioning input of a projection, the output selector is the empty set, e.g., in vertices 2, 4, and 5, as the sub-query matching only C events is a partitioning input for $p_1$.*

We now turn to the characterization of correctness and completeness of query evaluation with INEv graphs, i.e., we aim to ensure that all matches are generated at nodes hosting the workload queries.

Definition 2 (Valid INEv Graph). *Let $I = (V, E)$ be an INEv graph for a query workload $Q$ and an event network $\Gamma = (G_{top}, f, r)$ with the topology graph $G_{top} = (V_{top}, E_{top})$. $I$ is valid, if all of the following conditions are satisfied:*

*(1) The graph includes all queries: For each query $q \in Q$, there exists a vertex $v \in V$ that denotes a placement, i.e., $v_{proj} = q$.*

*(2) The graph induces combinations: For each vertex $v \in V$ and projection $v_{proj}$, the projections $\mathfrak{P}' = \{v_{proj}\} \cup \bigcup_{v' \in pre(v)} \{v'_{proj}\}$ with their inputs $\lambda'(p) = \{p' \in \mathfrak{P}' \mid \exists (v', v'') \in E \wedge v'_{proj} = p' \wedge v''_{proj} = p\}$ for all $p \in \mathfrak{P}'$, yield a combination, $(\mathfrak{P}', \lambda') \in C_{v_{proj}}$.*

*(3) The graph is source-closed: For a vertex $v \in V$ with leaf vertices $V_l = \{v' \in V \mid (v', v) \in E^* \wedge pre(v') = \emptyset\}$, let $\mathfrak{O}(v) \subseteq \mathcal{E} \times V_{top}$ be the pairs $(\epsilon, n)$ where event type $\epsilon$ is produced by $n$, $n \in f(\epsilon)$, and there is a leaf node $((O, \emptyset), n, \emptyset, \equiv) \in V_l$ where the primitive operator $o \in O_p$ references $\epsilon$. Then, $\mathfrak{O}(v)$ must include a pair $(\epsilon, n)$ for each event type $\epsilon \in \mathcal{E}(v_{proj})$.*

*(4) The graph covers all relevant sources: For a projection $p$ of a vertex $v \in V$, i.e, $v_{proj} = p$, let $\mathfrak{O}(p) = \{(\epsilon, n) \in \mathcal{E}(p) \times V_{top} \mid n \in f(\epsilon)\}$ be the set of all relevant event types and their source*

*nodes. There must exist a set of vertices $V_p \subseteq V$ that jointly cover all sources of event types of $p$, i.e., $\bigcup_{v \in V_p} \mathfrak{D}(v) = \mathfrak{D}(p)$.*

(5) *The graph is negation-closed: If a query $q = (O, \beta) \in \mathcal{Q}$ contains an NSEQ operator $o \in O$ with children $\beta(o) = \langle o_1, o_2, o_3 \rangle$, then for any projection $p = (O', \beta') \in \bigcup_{v \in V} \{v_{proj}\}$ used in $I$, it holds that $o_2 \in O'$ implies $o_1, o_3 \in O'$.*

(6) *The graph is Kleene-closed. If a query $q = (O, \beta) \in \mathcal{Q}$ contains a KL operator $o \in O$ with a primitive child $\beta(o) = \langle o_i \rangle$, then all vertices $V_{KL} \subseteq V$ in $I$ for which $o_i$ is the partitioning input of a multi-node placement have the same successor vertices in $I$.*

(7) *The graph captures complete forwarding. For each vertex $v \in V$ with predecessors, i.e., $pre(v) \neq \emptyset$, the following holds for all predecessors $u \in pre(v)$:*

   (i) *$v_{node}$ is contained in the set of nodes of $u_{graph}$.*

   (ii) *if $u_{os} \subset \mathcal{E}(v_{proj})$, then for all vertices $w \in V$ for which there exists an event type $\epsilon \in \mathcal{E}(w_{proj}) \cap \mathcal{E}(v_{proj}) \setminus u_{os}$, the node $v_{node}$ is contained in the set of nodes of $w_{graph}$.*

The above notion of a valid INEv graph implies correctness.

THEOREM 1. *For a workload $\mathcal{Q}$, the matches produced for query $q \in \mathcal{Q}$ by a node hosting $q$ in a valid INEv graph are matches of $q$.*

PROOF. (sketch) Consider a vertex $v$ of the INEv graph with $v_{proj} = q$. The graph induces combinations, condition (2), and ensures that negations are not split into different projections, condition (5). Hence, matches of $q$ can be assembled from matches of projections of predecessors in the graph, assuming that these matches are derived from all required events. The latter is ensured by the covering of sources of relevant event types by leaf nodes of $v$, condition (3), and the fact that matches from the nodes of $v$'s predecessors are forwarded to $v_{node}$, condition (7). The reasoning for the correctness of evaluating $q$ as part of $v$ based on the correctness of evaluating the projections at $v$'s predecessors may not be applied inductively, over all preceding vertices. □

A valid INEv graph ensures not only correctness, but also completeness. All matches of queries of the workload are generated.

THEOREM 2. *For a workload $\mathcal{Q}$, a valid INEv graph generates all matches of each query $q \in \mathcal{Q}$.*

PROOF. (sketch)

From condition (1) it follows, that a valid INEv graph contains at least one vertex $v$ with $v_{proj} = q$ for each query $q \in \mathcal{Q}$. Theorem 1 ensures correct evaluation of these queries. Yet, incompleteness may occur due to not all combinations of relevant event types being considered for a query, or due to not all events of a single type being matched by a Kleene operator. Though, these two cases are excluded by conditions (4) and (6), respectively. □

## 5.3 Transmission Cost Model

Turning to the efficiency of query evaluation with INEv graphs, we now consider the induced transmission costs. While the latter is given as the sum of the rates with which events are sent over the network per time unit, our cost model derives this total rate from the transmission costs per node in the event network. This cost depends on whether projections hosted by a node are subject to a single-node placement or a multi-node placement, to which nodes events are sent during query evaluation, and which output selectors are employed.

To compute the transmission cost for a vertex in an INEv graph, we incorporate an assumption on multi-node placements, as follows. Consider an INEv graph $I = (V, E)$ and a projection $p$ that is placed on multiple nodes, which is represented by the vertices $V_p \subseteq V$, such that for all $v \in V_p$ it holds that $v_{proj} = p$. Then, the partitioning input of the projection needs to be a projection that references a single event type, which must also be generated by the nodes of the respective vertices in $V_p$. Put differently, the INEv graph must contain vertices that place the partitioning input, a projection over a single event type, at the same nodes as the vertices in $V_p$. This assumption is motived by the general idea behind multi-node placements: Sending the partitioning input over the network may thwart the cost reduction aimed at with multi-node placements. In addition, considering more complex projections over those referencing a single event type will not lead to improvements, since any multiple-node placement of them could be exploited transitively.

EXAMPLE 7. *In Figure 3, there is a multi-node placement for projection $p = SEQ(A, C)$ with vertices 7-9, i.e., $V_p = \{7, 8, 9\}$. For $p$, a partitioning input referencing event type $C$ is used. Hence, there is a second set of vertices $\{2, 4, 5\}$ for this input, such that the nodes of these vertices correspond to the nodes of the vertices in $V_p$.*

For an INEv graph $I = (V, E)$, we define a function $tc : V \rightarrow \mathbb{R}$ that assigns transmission costs to the vertices. It is based on the *total rate* $R(p)$ of a projection $p$, i.e., the rate with which matches of $p$ are generated. If a vertex $v \in V$ does not contain any output selection ($v_{os} = \equiv$), matches of its projection $v_{proj}$ are sent over the forwarding graph $v_{graph}$ with $v$'s portion of the total rate $R(v_{proj})$.

Once output selection is adopted, however, we need to consider the selectivity of the projection per event type. We capture this aspect with the notion of a *single selectivity* $\sigma(\epsilon, p)$, which defines for an event type $\epsilon \in \mathcal{E}(p)$, the probability that an $\epsilon$ event is contained in a match of $p$. As such, the rate of distinctive $\epsilon$ events in matches of $p$ is given by $\sigma(\epsilon, p) \cdot R(\epsilon)$, which is always less than the rate $r(p)$ of the projection. However, in case of the partitioning input being part of the selected output, the rate with which events are sent depends on the local rate of the referenced event type, instead of the total rate, times the single selectivity. The intuition here is that nodes hosting a projection in a multi-node placement access only their locally generated events.

Integrating both cases, we arrive at the following definition of the cost function. As above, let $V_p \subseteq V$ denote the set of vertices of an INEv graph $I = (V, E)$ with their projection being $p$ and let $part(p)$ denote the (single) event type of the partitioning input of the projection. Then, for a vertex $v \in V$ with $v_{graph} = (V_v, E_v)$ as its forwarding graph, the transmission cost is defined as:

$$tc(v) = \begin{cases} |E_v| \cdot R(v_{proj}) \;/\; |V_p| & \text{If } v_{os} = \equiv. \\ |E_v| \cdot \sum\limits_{\epsilon \in v_{os} \setminus part(v_{proj})} \big(\sigma(\epsilon, v_{proj}) \cdot R(\epsilon)\big) + & Otherwise. \\ r(part(v_{proj})) \cdot \sigma(part(v_{proj}), v_{proj}) & \end{cases}$$

Based on the transmission costs per vertex, we derive the total transmission costs, incorporated in Problem 1. For an INEv graph

$I = (V, E)$ this cost is given as

$$\psi(I) = \sum_{v \in V} tc(v).$$

Example 8. *In Figure 3, the cost of vertex* 4 *is* 0*. Since C is the partitioning input of the multi-node placement of p, its rate does not add to the transmission costs. The cost of vertex* 3 *is given by the total rate of A (i.e., the local rate, as there is only one source of A events) multiplied by the number of edges of* $G_{A1}$*. Hence, the cost is* $11 \cdot r(A)$*. For projection* $SEQ(A, C)$ *with selected output C, vertices* 7-9 *have an output rate of* $r(C) \cdot \sigma(C, p)$*. As C is a partitioning input of* $SEQ(A, C)$*, the local rate is used.*

bsectionLatency Model Having discussed a model for the transmission cost, we now reflect on the latency of query evaluation, as it provides a constraining factor in-network evaluation, see Problem 1. As usual, the latency of a match of a query is the time between its occurrence and the occurrence of the latest event that led to the match's creation. It includes a processing latency inherent to query evaluation and a transmission latency observed between network nodes.

**Processing latency.** We model the latency induced by query evaluation at a particular node by the number of partial matches created and maintained. The reason being that for many query evaluation algorithms, whether they are based on automata [1] or operator trees [31], the time complexity depends on the number of maintained partial matches, which, in turn, depends on the query selectivity and rates of its inputs.

Let $I = (V, E)$ be an INEv graph and let $pm(p)$ be the rate with which partial matches are generated during the evaluation of a projection $p$. Note that if $p$ is evaluated with a multi-node placement, the rate $pm(p)$ is partitioned among the nodes of the placement. Hence, the processing latency induced by a vertex $v \in V$ is:

$$\ell_p(v) = \frac{pm(v_{proj})}{|\{u \mid u \in V \wedge u_{proj} = v_{proj}\}|}.$$

**Transmission latency.** The transmission latency of a match is given as the number of hops that need to be traversed in order to provide for the inputs necessary to generate the respective match.

Let $w, v \in V$ and let $w$ be a predecessor of $v$ in the INEv graph $I = (V, E)$. If $w_{node} \neq v_{node}$, i.e., the vertices describe placement at different network nodes, matches of $w_{proj}$ need to be forwarded to evaluate $v_{proj}$. The number of hops that need to be traversed is given by the length of the shortest path between the nodes $w_{node}$ and $v_{node}$ in the forwarding graph $w_{graph}$ of $w$.

Example 9. *In Figure 3, matches of* $q_2$ *are generated at node* 6 *(vertex* 10*) based on matches of* $SEQ(A, C)$ *generated at node* 10 *(vertex* 8*). Here, the transmission latency, i.e., the length of the shortest path between nodes* 6 *and* 10 *in the forwarding graph* $G_{A10}$*, is* 2*.*

Let $pre(v)$ denote the set of predecessors of $v$ and let $d(G, m, n)$ denote the length of the shortest path between the nodes $m$ and $n$ within the graph $G$. Based thereon, the transmission latency induced by a vertex $v \in V$ in an INEv graph $I = (V, E)$ is:

$$\ell_t(v) = \begin{cases} 0 & \text{If } pre(v) = \emptyset, \\ \max\{d(w_{graph}, v_{node}, w_{node}) \mid w \in pre(v)\} & \text{Otherwise.} \end{cases}$$

**Overall latency.** To model the overall latency, we combine the processing latency and the transmission latency in a weighted manner. While the former is influenced by the processing capacity of network nodes, the latter depends on properties of the underlying networking infrastructure. We therefore employ a factor $\xi \in \mathbb{R}$ to denote the ratio between the two sources of latency. Again, let $pre(v)$ denote the predecessors of a vertex $v \in V$ in an INEv graph $I = (V, E)$. Then, the latency for vertex $v$ is the processing latency, scaled by $\xi$, the maximal transmission latency from a predecessor, and the maximal overall latencies of the backwards closure of required inputs. Formally, the overall latency is defined inductively:

$$\ell_v(v) \begin{cases} 0 & \text{If } pre(v) = \emptyset, \\ \ell_p(v) \cdot \xi + \ell_t(v) + \max\{\ell_v(w) \mid w \in pre(v)\} & \text{Otherwise.} \end{cases}$$

Let $V_Q \subseteq V$ be the set of vertices of $I = (V, E)$ such that for each $v \in V_Q$, $v_{proj} \in Q$, i.e., the vertices in $V_Q$ denote the evaluation of the queries in $Q$. Then, the latency induced by $I$ is:

$$\ell(I) = \max\{\ell_v(v) \mid v \in V_Q\}.$$

## 5.4 Optimal INEv Graphs

Consider an event network $\Gamma$, a query workload $Q$, and a latency bound $\tau$, as defined in Problem 1 for in-network evaluation of CEP queries. Then, an INEv graph $I = (V, E)$ with $\ell(I) \leq \tau$ is optimal, if there exits no other INEv graph $I' = (V', E')$ with $\ell(I') \leq \tau$, such that $\psi(I') < \psi(I)$.

However, we note that computation of optimal INEv graphs is computationally hard. INEv graphs require the placement of the projections of a combination, which is given as a DAG. In [11], it was shown that operator placement for query graphs given as DAGs is NP-hard. Moreover, as INEv graphs support multi-node placements, the problem of connecting a set of nodes of a graph with a minimal sub-graph needs to be solved. This problem is known as the Steiner tree problem, a classical NP-hard problem [24].

# 6 CONSTRUCTION

Constructing optimal INEv graphs is generally intractable, so that we propose an approach that first constructs a combination for a query workload before determining the placement, output selectors, and forwarding subgraphs for each projection of the combination. Following preliminary considerations (§6.1), we introduce our algorithm to derive a combination for a single query (§6.2), explain how sharing opportunities between multiple queries are leveraged (§6.3), and present a placement and forwarding mechanism (§6.4). We also state the complexity of our algorithms (§6.5) and propose adaptation strategies to handle changes in the event network (§6.6).

## 6.1 Preliminary Considerations

**Suitable projections.** Before generating a combination for a query $q$, we determine all possible projections $\Pi_q$. For each projection $p \in \Pi_q$, we assess if $p$ can be used to reduce the transmission costs. This is the case when $p$'s global rate is smaller than the sum of the rates of $p$'s inputs. Over transitivity, this entails that $p$ is beneficial, if $R(p) > \sum_{\epsilon \in \mathcal{E}(p)} R(\epsilon)$. Intuitively, if the sum of the rates of $p$'s inputs are lower than the rate of $p$ itself, sending matches of $p$ results in higher transmission costs than sending matches of $p$'s inputs. Below,

we write $\Pi_{eval}$ for the set of projections that are suitable according to this condition.

**Steiner tree-based forwarding.** Due to multi-node placements and output selectors, INEv graphs contain vertices $v$ for which the selected output $v_{os}$ has to be forwarded to multiple nodes. To efficiently forward matches of $v_{os}$, a sub-graph of the topology graph that minimizes the number of edges shall be constructed. As mentioned, this requires to solve the Steiner tree problem and, thus, is intractable. We therefore employ a heuristic to construct Steiner trees based on Minimal Spanning Trees [28].

**Multi-node placements.** Let $c = (\mathfrak{P}, \lambda)$ be a combination. A multi-node placement shall be applied for projection $p$ of that combination, if there exists a partitioning input $i \in \lambda(p)$ with a much higher rate than all other inputs of $p$. Let $ST(N', G_{top})$ be a Steiner tree in our topology graph $G_{top}$ of a network $\Gamma = (G_{top}, f, r)$, which spans nodes $N'$. With $|ST(N', G_{top})|$ as the number of tree edges, we formulate a condition for the rate of the partitioning input $i$:

$$R(i) \cdot \delta > |ST(f(i), G_{top})| \cdot \sum_{p' \in \lambda(p) \setminus \{i\}} R(p') + R(p) \cdot \delta \qquad (1)$$

If $p \in Q$, matches of $p$ do not have to be sent over the network and the last summand can be omitted. The right part of Eq. 1 approximates the network costs caused by a multi-node placement at $i$, given that the inputs $\lambda(p)$ are forwarded using a Steiner tree. If forwarding all inputs of $p$ to the sources of $i$ results in less transmission cost than gathering events of type $i$ at some node (with cost $R(i) \cdot \delta$), a multi-node placement shall be used for $p$.

**Kleene closure evaluation.** To avoid superfluous data transmission, INEv graphs for workloads with Kleene closure materialize matches of these operators only at the query sink(s).

Technically, for a query $q_{KL} \in Q$ that comprises a Kleene closure operators, a query $q_{nKL}$ without these operators is derived by removing each Kleene operator $o_{KL}$ in $q_{KL}$ and assigning its child operator $\lambda(o_{KL})$ as a child to the parent operator of $o_{KL}$. For example, for the query $q_{KL} = SEQ(A, KL(B), C)$, the query $q_{nKL} = SEQ(A, B, C)$ is constructed and evaluated. The matches of $q_{KL}$ are then generated based on the matches of $q_{nKL}$ at the sinks.

**Time window-based scaling.** For an event network $\Gamma$, $r$ defines the number of occurrences per event type for a time unit $t$. To construct an INEv graph for a query $q$ having $w$ as time window, the event rates used as inputs are scaled by a factor $w/t$.

## 6.2 Combination Generation

Our approach to generate a combination is motivated as follows. Consider a combination $c$ of a query $q$ that contains only one projection $p$, such that neither $p$ nor $q$ yield a multi-node placement, i.e., none of the inputs $\lambda(p)$ and $\lambda(q)$ fulfil Eq. 1. Then, matches of all inputs of $p$ need to be sent to the node where $p$ is placed. Subsequently, matches of $p$ and the residual event types $\mathcal{E}(q) \setminus \mathcal{E}(p)$ are forwarded to the node where $q$ is placed. Such a combination with single-node placements yields little potential for saving transmission costs over a centralized model, as we later confirm empirically. Hence, we aim to maximize the cost savings induced by multi-node placements.

To this end, we use dynamic programming to enumerate combinations for each suitable projection and recursively construct a combination for the query. As shown in Alg. 1, we first initialize a combination $c$ with all projections in $\Pi_{eval}$ (line 1). Then, we iterate

**Algorithm 1:** Generation of a combination.

**input :** Event network $\Gamma$; query $q$; suitable projections $\Pi_{eval}$
**output :** Comb. $C_q = (\mathfrak{P}, \lambda)$; projections $MN \subset \mathfrak{P}$ for multi-node placement

1 $c = (\Pi_{eval}, \emptyset)$; `// Initialize combination`
2 **for** $p \in$ `topoSort`$(\Pi_{eval})$ **do** `// For each projection`
3 $Pre \leftarrow (\Pi_{eval} \cap \Pi_p) \cup \mathcal{E}(p)$ ; `// Possible inputs of p`
4 `getInputs`$(p, Pre, \emptyset)$;
5 **if** $\exists\, i \in \lambda(p)$ *for which Eq. 1 holds* **then** $MN \leftarrow MN \cup \{p\}$ ;
6 $c = (\{q\} \cup \lambda(q)^+, \lambda)$;
7 **return** $c$, $MN$;

`// Enumerate all possible sets of inputs for projection p`
8 **function** `getInputs`$(p, Pre, Inputs)$
`// Consider only set of inputs with all event types of p`
9 **if** $Pre = \emptyset \wedge \cup_{i \in Inputs} \mathcal{E}(i) = \mathcal{E}(p)$ **then**
10 **if** `getSavings`$(p, Inputs) <$ `getSavings`$(p, \lambda(p))$ **then**
11 $\lambda(p) \leftarrow Inputs$;
12 **else**
13 **for** $o \in Pre$ **do** `// For each predecessor`
14 $Pre' \leftarrow$ `prune`$(Pre \setminus \{o\}, Inputs \cup \{o\})$;
15 `getInputs`$(p, Pre', Inputs \cup \{o\})$;

over the projections based on a topological order, which ensures that all of projections $p$'s potential inputs have been handled before proceeding with $p$ (line 2). For each projection $p$, the set of possible inputs (line 3) is used to enumerate all possible combinations (line 4), thereby recursively determining the inputs that maximize the savings. Once the set of inputs maximizing the savings has been computed for each projection, multi-node placements are considered if possible (line 5) and the final combination is derived (line 6).

**Enumeration** (`getInputs`). The enumeration of all possible sets of inputs for projection $p$ proceeds recursively (line 8). The selection of input sets is guided by their estimated savings (line 10-line 11). Moreover, pruning strategies may exclude projections (line 14).

**Savings estimation** (`getSavings`). The savings of a projection and a set of inputs $I$ are estimated inductively. We add up the savings of all inputs, which, in turn, are derived from their inputs. If an input yields a multi-node placement, using Eq. 1, we add the savings induced by the multi-node placement. Otherwise, we reduce the savings by the costs of sending events of all inputs over $\delta$ hops, where $\delta$ is the average shortest path distance in the network.

We further incorporate two heuristics to avoid a certain bias in the estimation. First, we ensure that savings induced by a *single* partitioning input to *many* multi-node placements are added only once to the savings of a combination. Second, we add savings that originate from projections, for which, due to being input to multiple other projections with multi-node placements, the induced transmission costs can be shared among the target projections.

**Latency-based pruning** (`prune`). For each considered combination, we estimated the induced latency based on the processing latency and the transmission latency (estimated based on the combination depth). Then, we prune for each projection all combinations that violate the latency bound $\tau$ (see Problem 1).

## 6.3 Multi-Query Sharing

We apply Alg. 1 to the queries of a workload $Q$ to incrementally construct the combination $C_Q$. In this process, sharing opportunities between multiple queries are exploited, as follows.

**Shared construction.** Once the combinations derived for multiple queries share a projection, the results can be reused. Hence, the combination $C_q$ derived for a query $q$ by Alg. 1 is merged with the combinations for each previously processed query by unifying the respective sets of vertices and edges.

**Sharing-based scoring.** When constructing the combination for a workload iteratively, our two heuristics *sharedInputs* and *sharedMNSavings* always rely on the (partial) combination for the subset of queries $Q' \subseteq Q$ considered so far. As such, when considering the next query $q$, the inputs and multi-node placements of queries in $Q'$ are incorporated in the calculation of the savings of a combination for $q$, which fosters the reuse of projections between queries.

**Resolving conflicting multi-node placements.** Multi-node placements obtained for different queries may have the issue that the partitioning input of a placement for a first query, may not be a partitioning input of a placement for a second query. This is problematic, since all outputs of the partitioning input of the first query may need to be sent over the network to realize a multi-node placement for the second query. We avoid such *conflicting multi-node placements*, as follows. Once a query $q$ has been processed by Alg. 1, all non-partitioning inputs of the obtained combination are added to a set $\Theta \subset \mathcal{E}(q)$. When applying Alg. 1 to the next query, multi-node placements with partitioning inputs in $\Theta$ are excluded. By maintaining the set $\Theta$ for all queries, we avoid conflicting multi-node placements in the generated combination of the query workload.

The order in which Alg. 1 is applied to the queries of a workload influences the projections and multi-node placements of the INEv graph. However, as empirically shown in §7, the differences in the resulting costs for different orderings of the workload are negligible.

**Query workloads with varying time windows.** To evaluate a workload of queries with varying time windows, the event rates of the network for INEv graph construction are scaled with the largest time window of a query workload. Consequently, matches of each query must be filtered based on the original time window at the sinks.

## 6.4 Forwarding Graph and Placement

After a combination for the query workload has been generated, Alg. 2 constructs the forwarding graph and places the projections.

**Output selection.** First, the output selectors for each projection of the combination are determined. For each projection $p$ having a multi-node placement, all non-partitioning inputs of $p$ in $\mathcal{E}(p)$ are added to the set of disseminated events *DissEv* (line 2). As the inputs of multi-node placements have to be forwarded along at least as many hops as there are nodes of the multi-node placement, their dissemination in the network can be exploited for output selectors. For each projection $p$, we check if event types in $\mathcal{E}(p)$ are contained in the set of disseminated events *DissEv* (line 3). If so, for $p$, it suffices to select types not in *DissEv* as output. Yet, this is only beneficial if the rate of the potential selected output is less than the rate of the projection itself (line 26). Accordingly, the output selector $os(p)$ is determined and the rate of $p$ used for calculating the network transmission costs are updated.

Next, the set of inputs for each projection is extended regarding the output selectors (line 4). For each projection $p$ having an input $o \in \lambda(p)$ with $os(o) \subset \mathcal{E}(o)$, the input of $p$ is extended by $\mathcal{E}(o) \setminus os(o)$, i.e., the event types not contained in the selected output of $p$ are added to the inputs of $p$. This ensures that the forwarding graphs calculated in the next step are well-formed (see §5.2).

**Algorithm 2:** Forwarding graph construction & placement.

**input :** Event network $\Gamma = (G_{top}, f, r)$, combination $C_Q = (\mathfrak{P}, \lambda)$; projections $MN \subset \mathfrak{P}$ for multi-node placement
**output :** Vertices $V$ of INEv graph $I = (V, E)$

1 $costs$, $Successors \leftarrow \emptyset$ ;
2 **for** $p \in MN$ **do** $DissEv \leftarrow DissEv \bigcup (\lambda(p) \cap \mathcal{E}(p)) \setminus part(p)$;
3 **for** $p \in \mathfrak{P}$ **do** $os(p) \leftarrow$ `getOS`$(p, DissEv)$ ;
4 **for** $p \in \mathfrak{P}$ **do** $\lambda(p) \leftarrow \lambda(p) \bigcup \bigcup_{i \in \lambda(p)} \mathcal{E}(i) \setminus os(i)$ ;
`// For all inputs of projections with multi-node placements`
5 **for** $i \in \bigcup_{p \in MN} \lambda(p) \setminus part(p)$ **do**
6 $Successors[i] \leftarrow \{x \mid x \in f(part(p)) \wedge i \in \lambda(p)\})$ ;
7 **for** $p \in \mathfrak{P}$ **do** `// For each projection` $p$
8 **if** $p \in MN$ **then** `// If` $p$ `has a multi-node placement`
9 **for** $i \in \lambda(p)$ **do** `// For each input of` $p$
10 **for** $n \in f(i)$ **do** `// For each source of` $i$
11 $v_{graph} \leftarrow ST(Successors[i] \cup n, G_{top})$ ;
12 $V \leftarrow V \cup (i, n, v_{graph}, os(i))$ ;
13 **else** `// Only single-node placement exists for` $p$
14 **for** $m \in V_{top}$ **do** `// Consider each node for` $p$`'s placment`
15 **for** $i \in \lambda(p)$ **do** `// For each input of` $p$
16 **for** $n \in f(i)$ **do** `// For each source of` $i$
`// Add costs for sending` $i$ `from` $n$ `to` $m$
17 $costs[m] \leftarrow costs + SP(m, n) \cdot r(i)$ ;
18 $m \leftarrow \text{argmin}\, costs[m]$ `// Choose` $m$ `for placement of` $p$;
19 **for** $i \in \lambda(p)$ **do**
20 **for** $n \in f(i)$ **do**
21 **if** $v = (i, n, v_{graph}, os(i)) \in V$ **then**
22 $v_{graph} \leftarrow v_{graph} \cup SP(m, n)$;
23 **else** $V \leftarrow V \cup (i, n, SP(m, n), os(i))$;
24 **return** $V$
25 **function** `getOS` $(p, DissEv)$
26 **if** $\sum_{i \in \mathcal{E}(p) \setminus DissEv} R(i) \cdot \sigma_{(i,p)} < R(p)$ **then return** $\mathcal{E}(p) \setminus DissEv$;
27 **else return** $\equiv$;

**Shared placement.** If a projection $i$ is input to multiple other projections, the forwarding graph for $i$ is generated such that all of these projections are jointly considered. To this end, for each projection $i$, the set *Successors* is generated, which comprises the sources of event types that are partitioning input to a multi-node placement of a projection $p$ having $i$ as input (line 6).

Then, iterating over all projections, the vertices of an INEv graph are generated. If a projection $p$ has a multi-node placement, for each input $i \in \lambda(p)$ and each source $n \in f(i)$, a Steiner tree is constructed that spans $n$ and all nodes in *Successors* (line 11). The tree yields the forwarding graph $v_{graph}$ of the vertex $v$ having $i$ as $v_{proj}$ and $n$ as $v_{node}$. The vertex $v$ is added to the set $V$ (line 12).

If a projection $p$ has a single-node placement, we consider all nodes for $p$'s placement (line 14). We choose the one that minimizes the transmission costs using the shortest path length to calculate the costs of the placement (line 17). Let $m$ be the node chosen for a single-node placement of $p$ (line 18). For each input in $i \in \lambda(p)$ and each source $n \in f(i)$, it is checked if a vertex $v$ in $V$ exists having $v_{proj}$ and $v_{node}$ given by $i$ and $n$, respectively. If so, $v_{graph}$ is extended by the nodes and edges of the shortest path from $m$ to $n$ (line 22). Otherwise, a new vertex is added to $V$ (line 23).

**Final INEv graph construction.** Alg. 2 yields the vertices $V$ of the INEv graph $I = (V, E)$ for the combination constructed for a query workload. The set of edges $E$ contains all pairs of vertices $(v, w) \in V \times V$, for which $w_{proj} \in \lambda(v_{proj})$.

## 6.5 Complexity of Construction

Identifying suitable projections requires to enumerate the set of projections, which has a cardinality of $|2^{\mathcal{E}(q)}|$. During the combination generation, for each suitable projection, all combinations are enumerated, which yields a time complexity of $O(|2^{\mathcal{E}(q)}| \cdot 2^{|2^{\mathcal{E}(q)}|})$.

Given a combination, to derive the forwarding graph and placement, we iterate over all projections and all inputs of a projection, which, again, can be at most $|2^{\mathcal{E}(q)}|$ per projection. For each source $f(i)$ of an input $i$, at most the number of network nodes $|V|$, a Steiner tree is constructed with a heuristic that runs in $O(|E| \cdot \log |V|)$ time. As such, the second step of our construction has a total time complexity of $O(|2^{\mathcal{E}(q)}| \cdot |2^{\mathcal{E}(q)}| \cdot |V| \cdot |E| \cdot \log |V|)$.

## 6.6 Adaptive Repair

Once event rates and data distributions in an event network change over time, the quality of an INEv graph may degrade as its construction relies on network and query statistics. We therefore propose two strategies to adapt INEv graphs:

**Repairing invalid projections.** A projection becomes harmful, if its output rate becomes higher than the sum of its input rates. This situation can be locally detected by the network node hosting the respective projection. In this case, the node will forward the inputs of the projection directly instead of its matches.

**Repairing invalid multi-node placements.** For a projection to be beneficial for a multi-node placement, Eq. 1 has to hold. This is no longer the case, if the rate of the partitioning input decreases or the rates of the non-partitioning inputs of the placement increase. While the former does not add to the resulting network costs of the graph, the latter may incur significant overhead, as high-rate events have to be forwarded in the network excessively to all nodes of a multi-node placement. To mitigate this situation, for each multi-node placement, a (fall-back) single-node placement is computed during the INEv graph construction. Once a node locally detects the described situation, evaluation switches to the predefined single-node placement. All input sources and nodes of the, now invalid, multi-node placement are notified about the new single-node placement to forward their inputs respectively and evaluate the projection.

# 7 EXPERIMENTAL EVALUATION

We outline our setup for the evaluation of INEv graphs in §7.1. Results from a comparison of INEv graphs against state-of-the-art approaches for distributed CEP based on real-world and synthetic data are given in §7.2, before §7.3 reports on a sensitivity analysis.

## 7.1 Experimental Setup

Our implementation and experimental setup is publicly available.[1]

**Datasets and queries.** Our experiments relied on two real-world datasets. First, we used one month of the Citi Bike dataset [20] that captures information about bike rentals. It contains 1 million events, each of them describing the duration of a trip, the start and end station, a bike ID, and information about the driver. We considered various partitions of the dataset to investigate networks of different sizes. Specifically, the data was split based on the end station of a trip to obtain datasets for 5, 10, 20, 30, or 50 nodes in a network. The network structure was selected randomly with an average out-degree of three. We further derived nine event types characterizing bike trips of varying duration. Based thereon, we designed four queries (available online) describing different usage scenarios for the same bike over a 24 hours time window.

As a second dataset, we employed the Google Cluster traces [47], which comprise around 4.5 million events. Each event contains attributes, such as an execution priority, a user ID, the ID of the machine on which the task shall be executed. The latter attribute was used to partition the dataset for 5, 10, 20, 30, or 50 nodes in a network of random structure (average out-degree of three). Based on the task status, we obtained nine event types. Again, we used four queries (available online) describing common monitoring scenarios. Based thereon, we derived the selectivities for a time window of 30 minutes which reflects the life-time of the majority of jobs.

We further used synthetic data to achieve a controlled setup. We generated event networks varying in their number of nodes, event types, distribution of event rates (*event skew*), the portion of event types generated per node (*event node ratio*), and the out-degree per node. For the event skew, we draw rates from a Zipfian distribution. If not stated otherwise, we used an event skew of 1.3, a network of 20 nodes, an event node ratio of 0.5, and an average out-degree of three. We constructed workloads that vary in their number of queries, query length, number of $KL$ and $NSEQ$ operators, event types, and selectivity scale, i.e., the range from which the selectivities are drawn uniformly. Here, a scale of 0.1 means that the predicate selectivities are between $1 - 10\%$. Single selectivities of event types are drawn uniformly from predicate selectivities.

**Baselines.** We evaluated INEv graphs against a centralized baseline that does not employ query splitting and gathers all events required for the evaluation a query workload at one node. Here, we chose the node minimizing transmission costs.

We also considered two state-of-the-art approaches for distributed CEP: In [23], single-node operator placement is combined with push-pull communication. The idea of push-pull-based CEP is to send (push) only events of types with low rates. These events may then trigger a request to pull events of high-rate types from their sources. Thus, push-pull communication, similar to a multi-node placement, benefits from skewed event rates. For the comparison, we used an exhaustive search algorithm presented in [23] that computes optimal push-pull operator placements. MuSE [4], in turn, employs query rewriting and multi-node placements. A respective evaluation plan defines a set of query projections together with a placement at network nodes. Unlike INEv graphs, however, the model ignores the network topology. We therefore adopted forwarding of events based on shortest paths in the evaluation.

Finally, we considered query evaluation with Apache Flink [13], a state-of-the-art engine for distributed stream processing (DSP). However, being a DSP engine, Flink distributes the data explicitly in a network for query evaluation, neglecting where the data is generated. In our experiments, we therefore used an attribute of the respective query as a partitioning key to achieve distributed

[1] https://anonymous.4open.science/r/INEv-7F88/README.md

**Table 3**

Comparison against the state of the art in terms of the transmission ratio using real-world datasets.

| | | Citi Bike | | | | Cluster Monitoring | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Q1 | Q2 | Q3 | Q4 | Q1 | Q2 | Q3 | Q4 |
| 5 | INEv | 0.002 | 0.002 | 0.13 | 0.01 | 0.008 | 0.003 | 0.0004 | 0.005 |
| | MuSE | 0.57 | 0.003 | 0.15 | 1.5 | 0.06 | 0.4 | 0.02 | >2.2 |
| | PPoP | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 10 | INEv | 0.001 | 0.009 | 0.008 | 0.01 | 0.001 | 0.002 | 0.001 | 0.003 |
| | MuSE | 0.2 | 1.4 | 2.1 | 0.5 | 0.81 | 0.74 | 0.002 | 0.05 |
| | PPoP | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 20 | INEv | 0.007 | 0.05 | 0.001 | 0.008 | 0.006 | 0.008 | 0.2 | 0.009 |
| | MuSE | 0.03 | 1.5 | 1.2 | > 1.3 | >0.5 | 0.11 | 0.001 | 0.008 |
| | PPoP | 0.78 | 1 | 0.78 | 0.78 | 1 | 1 | 1 | 1 |
| 30 | INEv | 0.001 | 0.02 | 0.006 | 0.01 | 0.0007 | 0.006 | 0.0008 | 0.0003 |
| | MuSE | 0.27 | 1.8 | 2.7 | 1.7 | >0.65 | >0.62 | 0.0008 | 0.06 |
| | PPoP | 0.52 | 0.9 | 0.52 | 0.52 | 0.9 | 0.82 | 1 | 0.82 |
| 50 | INEv | 0.002 | 0.42 | 0.33 | 0.02 | 0.006 | 0.007 | 0.0005 | 0.03 |
| | MuSE | 0.26 | 1.1 | 1.6 | 0.07 | >0.5 | >0.45 | 0.002 | 0.04 |
| | PPoP | 0.37 | 0.66 | 0.37 | 0.37 | 0.66 | 0.43 | 1 | 0.43 |
| 10 | Flink | 1.6 | 1.4 | 2.1 | 1.6 | 2.5 | 1.4 | 4.1 | 3.6 |

evaluation. Again, events were disseminated in the network adopting shortest paths.

**Metrics.** We assessed the network transmission costs using the costs of centralized evaluation as a reference point: The *transmission ratio* is the cost induced by any of the approaches relative to the cost of the centralized evaluation. Moreover, we report *latency* and *throughput* distributions (median, quartiles) for query evaluation. For INEv graphs, we also measured the *computation time* for the construction of an evaluation plan.

**Implementation.** Our experiments with real-world data as well as all experiments to measure latency and throughput employed an automata-based CEP engine, written in C#. It supports the generation of partial outputs as defined by output selectors and handles the out-of-order arrival of events caused by the distributed evaluation. For experiments with 10 network nodes (i.e., 10 data partitions), the engine was deployed on a cluster of 10 Raspberry Pi 4B nodes (4GB RAM), running Raspbian GNU/Linux 10. The nodes of the cluster use TCP sockets for communication. Experiments with other network sizes relied on a NUMA node with 4 Intel Xeon E7-4880 (60 cores, 1TB RAM), using Unix sockets for the communication.

For the sensitivity analysis based on synthetic data, we used a Python implementation of our algorithms. The results are averaged over 50 runs per experiment.

## 7.2 State-of-the-art Comparison

**Transmission costs (real-world data).** Table 3 shows the transmission ratios (the lower, the better) induced by query evaluation with INEv graphs, MuSE [4, 5] and operator placement with push-pull (PPoP) [23] for networks of sizes 5 to 50.

First, INEv graphs generally yield very low transmission ratios, mostly $\leq 0.05$. This indicates a large reduction in the data transmission that is achieved by in-network evaluation. We attribute this observation to the relatively high selectivities of the queries (around 0.02 for Citi Bike dataset). INEv graphs leverage these selectivities, as they offer low output rates of projections, which can be used to construct multi-node placements. However, we observe relatively large ratios for Citi Bike and a network of 50 nodes. Here, the large networks lower the local rates of event types, which eventually limits the ability to find partitioning inputs for multi-node placements.

MuSE achieves comparable results, especially for the network of size 5, as it also employs multi-node placements. Yet, MuSE does not employ output selection and explicit forwarding, resulting in overall higher transmission ratios. As it also ignores the network topology in placement decisions, transmission costs may even exceed those of a centralized evaluation. Note that for some cases, we include a lower bound as query evaluation did not terminate within 24 hours due to overload caused by high network costs.

PPoP, in turn, degrades to centralized evaluation, when the local rate of all relevant event types exceeds one. This is the case for all queries for network sizes 5 and 10. For larger networks, the local rates decrease, which offers more optimization potential for PPoP. Yet, even for 50 nodes, is higher than for INEv graphs.

Table 3 also includes the transmission ratio of the distributed evaluation with Flink for a network of 10 nodes. As the attribute-based scheme used to distribute query evaluation is unaware of the network topology, there is no benefit in terms of transmission costs.

**Transmission costs (synthetic data).** Figure 4a-Figure 4b show the transmission ratios obtained for synthetic data for INEv graphs and PPoP under varying network parameters, for a query over 6 event types and a network of 20 nodes. Figure 4a considers a case where the number of events within a time window for each event type is $\leq 1$. Here, PPoP and INEv graphs show similar performance. Yet, if events of a certain type occur more than once within the time window of the query, considered in Figure 4b, *PPoP* degrades to a single-node placement, which shows a negligible reduction in transmission costs compared to centralized evaluation. INEv graphs still achieve a significant reduction in transmission costs.

For a query over 6 event types, INEv graphs also outperform MuSE, see Figure 4c-Figure 4d by up to 2 orders of magnitude. This is especially pronounced for small event skews and larger networks. Here, the number of hops over which events are forwarded rises and thus the optimization potential of output selection and Steiner Tree-based forwarding increases.

Moreover, Figure 4e shows the closeness to the lower bound for a query containing 6 event types, when varying the event skew and the selectivity scale (0.1 or 0.01). As computing optimal INEv graphs quickly becomes intractable, we approximated a lower bound for the transmission costs caused by an optimal INEv graph as follows: For each event type $\epsilon$ referenced in a query $q$, we check if there exists a projection $p \in \Pi_q$ for which either $R(p) < r(\epsilon)$ or $\sum_{\epsilon' \in \mathcal{E}(p)} R(\epsilon') < r(\epsilon)$. If so, we assume that an optimal solution comprises a combination in which $\epsilon$ is a partitioning input of a multi-node placement and, thus, events of this type are not sent over the network. Let $cand(q)$ denote the set of event types, for which the above holds true. Then, $\sum_{\epsilon \in \mathcal{E}(q) \setminus cand(q)} (R(\epsilon) - r(\epsilon)) \cdot \delta$ serves as a lower bound of transmission costs, assuming that all event types that are not partitioning inputs have to be forwarded over at least $\delta$ hops. Our algorithms approximate the lower bound well and yield close-to-optimal INEv graphs for skewed event rates.

**Latency and throughput (real-world data).** Figure 5 shows the latency and throughput observed for query evaluation on the Citi Bike dataset for INEv graphs (I) and Flink (F) on our Raspberry Pi cluster. Due to network overload, MuSE-based query evaluation did not produce any query matches and could thus not be included.

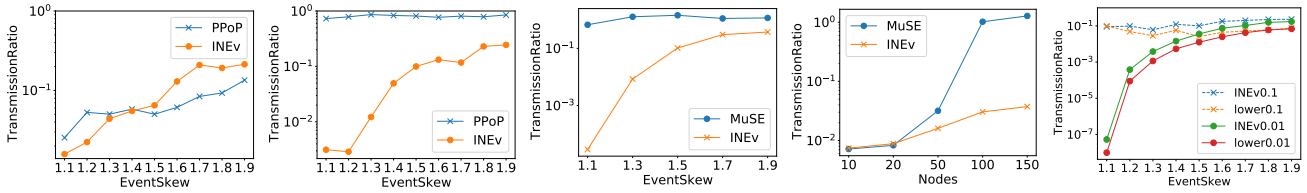



**Figure 4: Comparison against the state of the art and closeness to lower bound based on synthetic data.**

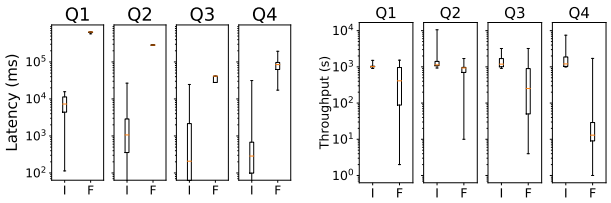



**Figure 5: Latency and throughput of INEv graphs (I) and Flink (F) for real-world data. MuSE is not included as no matches could be found due to network overload.**

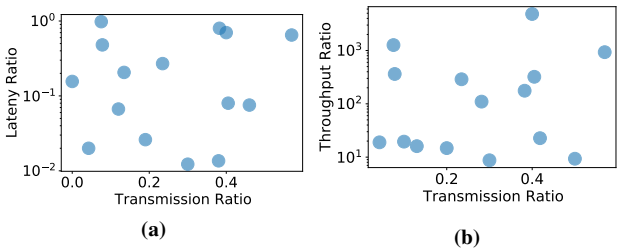



**Figure 6: Latency and throughput for synthetic data.**

INEv graphs reduce the latency by almost 2 orders of magnitude ($Q4$) compared to Flink, which suggests that a reduction in the transmission ratio (see Table 3) favours latency improvements. In terms of throughput, INEv graphs yield improvements by up to 2 orders of magnitude over Flink and also show less variance.

**Latency and throughput (synthetic data)** Figure 6 illustrates the latency and throughput of INEv graphs relative to centralized evaluation based on synthetic data. Here, we evaluated 15 INEv graphs for different workloads and event rate distributions, again on our Raspberry Pi cluster. Figure 6a shows the ratio of the latency (lower is better) of INEv graphs inducing varying transmission ratios. INEv graphs reduce detection latency by up to 2 orders of magnitude, especially for plans with low transmission ratios. The reason is that with low transmission costs, the number of events to process per node also decreases, leading to fewer partial matches per node. Similar results were observed in our throughput experiment (Figure 6b).

## 7.3 Sensitivity Analysis

**Network parameters.** Figure 7 shows the transmission ratios for a single query that references 7 event types, where line types represent different scales for the predicate selectivities. Considering the network parameters, the event skew impacts the quality of INEv graphs the most (Figure 7a), leading to a reduction in transmission costs of up to eight orders of magnitude compared to a centralized evaluation. For larger network sizes and event node ratios (Figure 7b and Figure 7c), the number of sources per event type increases, leading to a reduced optimization potential for lower selectivities. As the global rates of the event types rise, less suitable projections for the evaluation can be found, and thus, fewer multi-node placements become possible.

The out-degree of nodes has little impact on the quality of INEv graphs (Figure 7d). For the selectivity range $1 - 10\%$, INEv graphs benefit from a small out-degree, since longer network paths lead to higher event dissemination that can be leveraged by output selectors.

**Multi-query setting.** Figure 8 shows the quality of INEv graphs for workloads comprising 5, 10, and 20 queries over 6 event types, with a third of the workload queries comprising Kleene closure and $NSEQ$ operators. We used selectivities between $0.1 - 1\%$ for this experiment. As discussed in §6.3, increasing the overlap in event types between the queries increases the number of conflicting multi-node placements. Figure 8a illustrates this effect. For 20 queries over 6 event types, due to conflicting multi-node placements, the INEv graphs degrade to a centralized model. Yet, workloads of sizes 5 and 10 still show improvements in the transmission ratio. Increasing the number of event types, the transmission ratios improve significantly. Figure 8b, Figure 8c, and Figure 8d illustrate the sensitivity of the multi-query setup when considering 25 event types. The results confirm the trends obtained for the single-query scenario.

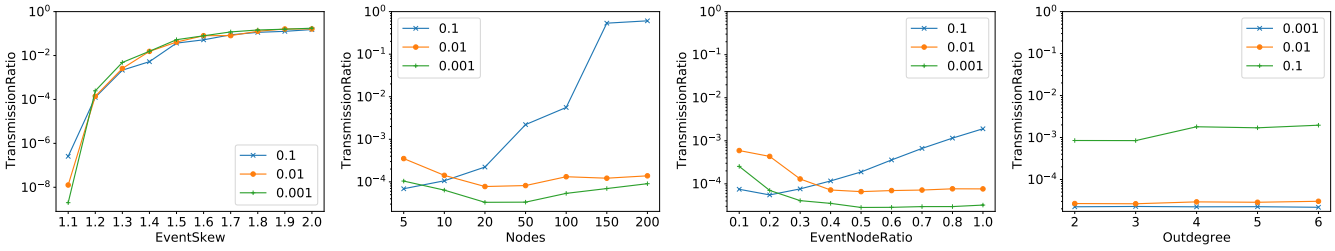
**Influence of Kleene and negation.** To study the impact of Kleene closure and negation, we generated two queries with the respective operator and derived counterparts without the operators, as follows.

For Kleene closure, we transformed the respective query as described in §6.1. For the negation, we replaced the $NSEQ$ operator with a $SEQ$ operator. For each set of queries, we assigned the child of the Kleene closure operator, or the negated operator of the $NSEQ$ operator, respectively, the event type having the highest ($max\_KL/NSEQ$) or lowest ($min\_KL/NSEQ$) rate.

Figure 9a shows that the Kleene closure operator has no impact on the transmission costs. Figure 9b indicates that the $NSEQ$ operator restricts the number of projections considered for combination generation, leading to potentially higher transmission costs.

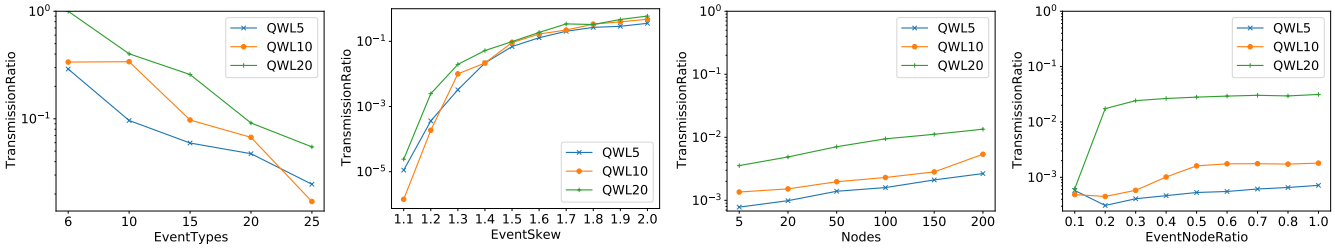
**Benefit of sharing.** To study how our combination generation and placement strategies leverage sharing opportunities, we generated workloads of sizes 5, 10, and 20 based on 10 event types. Figure 10a shows the ratio of transmission costs of our approach compared to a sharing-agnostic solution that treats each query in isolation (lower is better). Our approach reduces the transmission ratio, on average, by 30% and 80% for workloads of size 5 and 20, respectively.

Next, we assess the impact of the order in which the queries are processed during the incremental combination generation. For each workload size (between 5 and 30), we considered 50 workloads and for each workload, 50 different orderings of queries. Figure 10b



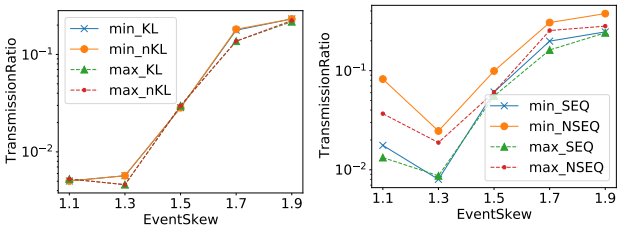

(a) (b) (c) (d)

**Figure 7: Transmission ratio of INEv graphs (lower is better) under varying network parameters and selectivity ranges.**



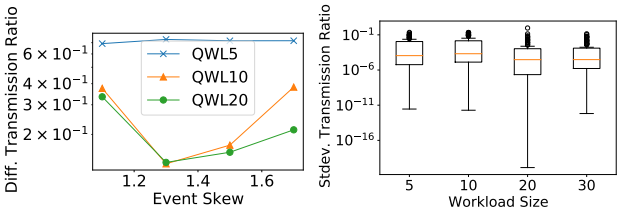

(a) (b) (c) (d)

**Figure 8: Transmission ratio for multi-query scenario for different workload sizes.**



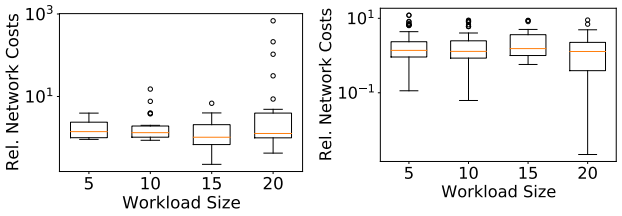

(a) (b)

**Figure 9: Influence of Kleene closure and negation.**




(a) (b)

**Figure 10: Incremental combination generation.**




(a) (b)

**Figure 11: Time window differences.**

reports the standard deviation of transmission ratios of INEv graphs based on combinations generated with different orderings. The standard deviation turns out to be small, i.e., $< 0.001$, suggesting that the order in which queries are processed has a negligible impact on the quality of the resulting combination. This result indicates a certain robustness when new queries are incorporated over time, as a complete re-computation of the combination may not be required.

Finally, we study the influence of queries for which the time windows vary by up to $\times 5$ ( Figure 11a) and $\times 10$ (Figure 11b). We report on the ratio of the transmission cost with an INEv graph for the complete query workload, and the aggregated costs induced by evaluating an INEv graph for each query of the workload in isolation. For both time window ranges, the results are close to 1, which indicates that evaluating all queries over the same time window does not significantly harm the quality of the resulting INEv graph. Also, larger networks show better ratios, which is due to increased result sharing. However, while for time windows from a smaller range (Figure 11a), shared evaluation almost always improves costs (values $< 1$), this is not the case for the large range (Figure 11b). To mitigate the latter, one may partition the workload according to the time windows of the queries and construct INEv graphs for these partitions accordingly.

**Adaptivity.** In Figure 12, we report on the effect of network changes on the quality of INEv graphs. For Figure 12a – Figure 12d, we considered a query workload of 15 queries, over 20 event types, and successively swapped their rates. We ordered the event types by their rates and stepwise increased the number of them having their rates swapped starting in the middle: For 0.1 *swaps*, the rates of the types having the $10th$ and $11th$ highest rates are swapped; and for 1.0 *swaps* the rates of all event types are swapped. We considered a selectivity range of 0.1 (Figure 12a, Figure 12b) and 0.001 (Figure 12c, Figure 12d). Figure 12a and Figure 12c report on the portion of projections and multi-node placements of the initial INEv graph (0 *swaps*) becoming invalid (see §6.1). The associated transmission ratios (*Broken*), and those resulting from the adaptation strategies proposed in §6.6 (*Repair*), are shown in Figure 12b and Figure 12d. Here, the transmission costs deteriorate with increasingly

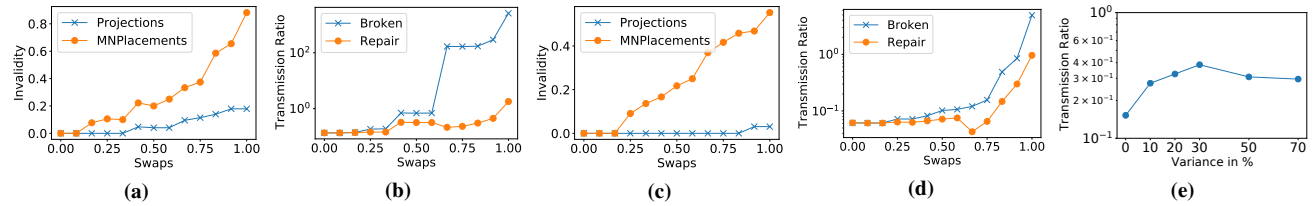



**Figure 12: Experiments on the adaptivity of INEv graphs to changes in the event network.**

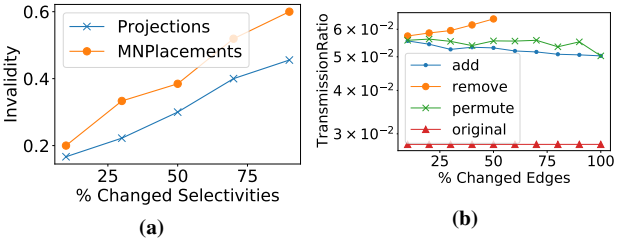



**Figure 13: Changes in selectivity and network topology.**

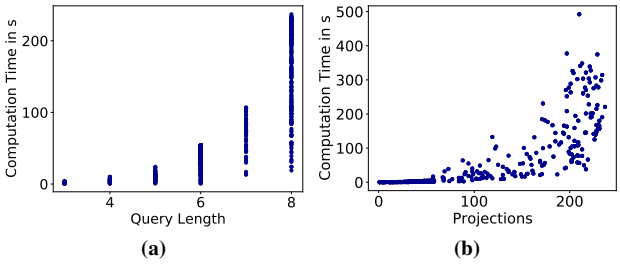



**Figure 14: Computation time.**

severe changes in the network. However, our adaptation strategy largely mitigates this effect.

In Figure 12e, we investigated the impact of variance in the event generation, with $x$% denoting that a variance of $x$% of the original rate can be observed over the duration of the experiment. The transmission ratios generally increase when the actual rates deviate from the estimates used for the respective INEv graph construction. However, for all considered variances, the resulting transmission ratios stay within a tight interval and only increase by at most 2.5×.

Furthermore, changed selectivities directly translate into changed rates of projections, which may invalidate them as well as related multi-node placements. To investigate this effect, we stepwise increased a percentage of the predicate selectivities by ×10. Figure 13a shows that the transmission ratio increase, when the query becomes less selective. To counter this effect, the strategy discussed above for changes in network rates can be applied.

Lastly, we consider changes in the topology of the network graph. Here, we examine three scenarios: edges of the graph may be added, removed, or permuted. If such a change occurs, forwarding as described by an INEv graph becomes obsolete, as the forwarding graphs may no longer exist. In that case, shortest path-based forwarding (computed with a distributed algorithm, such as distance-vector routing [46]) can be employed. In Figure 13b, we see that the transmission ratio increases when edges are removed, as the distances between nodes increase. With 50% of the edges removed, no connected graph can be constructed. While permutations of the edges do not show a specific trend, adding edges to the graph improves transmission costs, as the distances between nodes decrease.

**Plan generation efficiency.** Figure 14 illustrates the time needed for the computation of INEv graphs. The runtime largely depends on the number of projections considered during the combination generation, which increases the search space exponentially, as shown in Figure 14 . The number of projections, in turn, depends on the query length as well as the predicate selectivities. However, due to our pruning strategies, we can handle queries over 8 event types (Figure 14a) based on up to 230 projections, within an acceptable time.

## 8 RELATED WORK

**CEP optimization.** Optimized evaluation of CEP queries may leverage load shedding [15, 48], prefetching [49], result sharing [12, 36], or parallelization [10]. Moreoover, to maximize result sharing, query rewriting based on arbitrary projections was proposed in [27], but only for a centralized setting where data transmission costs do not occur. Query rewriting to reduce intermediate results in distributed CEP was considered in [41]. Unlike INEv graphs, however, this work neglects the optimization potential induced by multi-node placements and the network's topology. Push-pull communication in CEP [3] aims at reducing transmission costs by leveraging skewed event rates, as done in INEv graphs. However, the model of [3] does not consider in-network processing, rendering it topology-agnostic.

**In-network processing.** In-network processing was investigated for relational stream processing [2, 16, 32, 34, 42] and for CEP [17, 21, 30]. While these approaches exploit the network topology for placement decisions, no rewriting or splitting of queries based on the given topology or distribution of event sources is considered. In [23], operator placement was extended with the aforementioned push-pull paradigm [3]. While the proposed placement strategy incorporates operator reuse among queries, no prior rewriting is applied to enhance sharing opportunities. Distributed CEP with multi-node placements was addressed in [4, 5] for clique event networks. INEv graphs, in turn, can handle any network topology.

**Distributed stream joins.** The reduction of network transmission has been explored for distributed evaluation of continuous relational queries [29, 35, 39, 40, 50]. For instance, selective broadcasting may reduce data transmission in distributed stream join processing [35, 39], which exploits skewed data distributions. Yet, query rewritings to foster selective broadcasts are not considered. All of the above approaches target a single query and ignore the network topology.

**Distributed databases.** Our setting resembles horizontally fragmented relations in distributed databases [33], but without fragmentation rules that could be exploited for query optimizations. Semi-join reducers [6, 18, 19] are similar to our output selectors, which also restrict the results to be sent over the network. However, in INEv graphs, the data disseminated already in the network is incorporated.

# 9 CONCLUSIONS

We proposed INEv graphs as a model for in-network evaluation of CEP queries. INEv graphs define how to split queries, place them at network nodes, and forward partial results, thereby fostering use of disseminated events in the network to reduce transmission costs. We introduced algorithms for the construction of INEv graphs that exploit sharing opportunities in multi-query settings. Our evaluation shows that our algorithms yield close-to-optimal results and reduce transmission costs by up to eight orders of magnitude.

# REFERENCES


[1] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient pattern matching over event streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, Jason Tsong-Li Wang (Ed.). ACM, 147–160. https://doi.org/10.1145/1376616.1376634

[2] Yanif Ahmad and Ugur Cetintemel. 2004. Network-aware query processing for stream-based applications. In *Proceedings of the International Conference on Very Large Data Bases*. 456–467.

[3] Mert Akdere, UÇğur Çetintemel, and Nesime Tatbul. 2008. Plan-based complex event detection across distributed sources. *Proceedings of the VLDB Endowment* 1, 1 (2008), 66–77.

[4] Samira Akili and Matthias Weidlich. 2021. MuSE Graphs for Flexible Distribution of Event Stream Processing in Networks. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 10–22. https://doi.org/10.1145/3448016.3457318

[5] Samira Akili and Matthias Weidlich. 2021. Reasoning on the Efficiency of Distributed Complex Event Processing. *Fundam. Informaticae* 179, 2 (2021), 113–134. https://doi.org/10.3233/FI-2021-2017

[6] Peter M. G. Apers, Alan R. Hevner, and S. Bing Yao. 1983. Optimization algorithms for distributed queries. *IEEE transactions on software engineering* 1 (1983), 57–68.

[7] Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom. 2016. STREAM: The Stanford Data Stream Management System. In *Data Stream Management - Processing High-Speed Data Streams*, Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi (Eds.). Springer, 317–336. https://doi.org/10.1007/978-3-540-28608-0_16

[8] Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. 2017. Complex Event Recognition Languages: Tutorial. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017*. ACM, 7–10. https://doi.org/10.1145/3093742.3095106

[9] Alexander Artikis, Matthias Weidlich, François Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann, Katharina Morik, Vana Kalogeraki, Jakub Marecek, Avigdor Gal, Shie Mannor, Dimitrios Gunopulos, and Dermot Kinane. 2014. Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*, Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy (Eds.). OpenProceedings.org, 712–723. https://doi.org/10.5441/002/edbt.2014.77

[10] Cagri Balkesen, Nihal Dindar, Matthias Wetter, and Nesime Tatbul. 2013. RIP: run-based intra-query parallelism for scalable complex event processing. In *The 7th ACM International Conference on Distributed Event-Based Systems, DEBS '13, Arlington, TX, USA - June 29 - July 03, 2013*, Sharma Chakravarthy, Susan Darling Urban, Peter R. Pietzuch, and Elke A. Rundensteiner (Eds.). ACM, 3–14. https://doi.org/10.1145/2488222.2488257

[11] Shahid H. Bokhari. 1981. A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System. *IEEE Trans. Software Eng.* 7, 6 (1981), 583–589. https://doi.org/10.1109/TSE.1981.226469

[12] Lei Cao, Jiayuan Wang, and Elke A. Rundensteiner. 2016. Sharing-Aware Outlier Analytics over High-Volume Data Streams. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 527–540. https://doi.org/10.1145/2882903.2882920

[13] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.* 38, 4 (2015), 28–38. http://sites.computer.org/debull/A15dec/p28.pdf

[14] Sharma Chakravarthy and D. Mishra. 1994. Snoop: An Expressive Event Specification Language for Active Databases. *Data Knowl. Eng.* 14, 1 (1994), 1–26. https://doi.org/10.1016/0169-023X(94)90006-X

[15] Koral Chapnik, Ilya Kolchinsky, and Assaf Schuster. 2021. DARLING: Data-Aware Load Shedding in Complex Event Processing Systems. *Proc. VLDB Endow.* 15, 3 (2021), 541–554. http://www.vldb.org/pvldb/vol15/p541-chapnik.pdf

[16] Georgios Chatzimilioudis, Alfredo Cuzzocrea, Dimitrios Gunopulos, and Nikos Mamoulis. 2013. A novel distributed framework for optimizing query routing trees in wireless sensor networks via optimal operator placement. *J. Comput. System Sci.* 79, 3 (2013), 349–368.

[17] Jianxia Chen, Lakshmish Ramaswamy, David K Lowenthal, and Shivkumar Kalyanaraman. 2012. Comet: Decentralized complex event detection in mobile delay tolerant networks. In *2012 IEEE 13th International Conference on Mobile Data Management*. IEEE, 131–136.

[18] Ming-Syan Chen and Philip S Yu. 1992. Interleaving a join sequence with semijoins in distributed query processing. *IEEE Transactions on Parallel and Distributed Systems* 3, 5 (1992), 611–621.

[19] M-S Chen and Philip S. Yu. 1993. Combining joint and semi-join operations for distributed query processing. *IEEE Transactions on Knowledge and Data Engineering* 5, 3 (1993), 534–542.

[20] citi Bike. 2022. http://www.citibikenyc.com/system-data..

[21] Gianpaolo Cugola and Alessandro Margara. 2013. Deployment strategies for distributed complex event processing. *Computing* 95, 2 (2013), 129–156.

[22] Raul Castro Fernandez, Matthias Weidlich, Peter R. Pietzuch, and Avigdor Gal. 2014. Scalable stateful stream processing for smart grids. In *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014*, Umesh Bellur and Ravi Kothari (Eds.). ACM, 276–281. https://doi.org/10.1145/2611286.2611326

[23] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Network-wide complex event processing over geographically distributed data sources. *Inf. Syst.* 88 (2020). https://doi.org/10.1016/j.is.2019.101442

[24] M. R. Garey and David S. Johnson. 1977. The Rectilinear Steiner Tree Problem in NP Complete. *SIAM Journal of Applied Mathematics* 32 (1977), 826–834.

[25] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Complex event recognition in the Big Data era: a survey. *VLDB J.* 29, 1 (2020), 313–352. https://doi.org/10.1007/s00778-019-00557-w

[26] Mohit Jain, Vikas Chandan, Marilena Minou, George A. Thanos, Tri Kurniawan Wijaya, Achim Lindt, and Arne Gylling. 2015. Methodologies for effective demand response messaging. In *2015 IEEE International Conference on Smart Grid Communications, SmartGridComm 2015, Miami, FL, USA, November 2-5, 2015*. IEEE, 453–458. https://doi.org/10.1109/SmartGridComm.2015.7436342

[27] Ilya Kolchinsky and Assaf Schuster. 2019. Real-Time Multi-Pattern Detection over Event Streams. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 589–606. https://doi.org/10.1145/3299869.3319869

[28] Lawrence T. Kou, George Markowsky, and Leonard Berman. 1981. A Fast Algorithm for Steiner Trees. *Acta Informatica* 15 (1981), 141–145. https://doi.org/10.1007/BF00288961

[29] Qian Lin, Beng Chin Ooi, Zhengkui Wang, and Cui Yu. 2015. Scalable distributed stream join processing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 811–825.

[30] Manisha Luthra, Boris Koldehofe, Niels Danger, Pascal Weisenburger, Guido Salvaneschi, and Ioannis Stavrakakis. 2021. TCEP: Transitions in operator placement to adapt to dynamic network environments. *J. Comput. Syst. Sci.* 122 (2021), 94–125. https://doi.org/10.1016/j.jcss.2021.05.003

[31] Yuan Mei and Samuel Madden. 2009. ZStream: a cost-based query processor for adaptively detecting composite events. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul (Eds.). ACM, 193–206. https://doi.org/10.1145/1559845.1559867

[32] Matteo Nardelli, Valeria Cardellini, Vincenzo Grassi, and Francesco LO PRESTI. 2019. Efficient Operator Placement for Distributed Data Stream Processing Applications. *IEEE Transactions on Parallel and Distributed Systems* (2019).

[33] M Tamer Özsu and Patrick Valduriez. 1996. Distributed and parallel database systems. *ACM Computing Surveys (CSUR)* 28, 1 (1996), 125–128.

[34] Peter R. Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo I. Seltzer. 2006. Network-Aware Operator Placement for Stream-Processing Systems. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang (Eds.). IEEE Computer Society, 49. https://doi.org/10.1109/ICDE.2006.105

[35] Orestis Polychroniou, Rajkumar Sen, and Kenneth A Ross. 2014. Track join: distributed joins with minimal network traffic. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1483–1494.

[36] Olga Poppe, Chuan Lei, Lei Ma, Allison Rozet, and Elke A. Rundensteiner. 2021. To Share, or not to Share Online Event Trend Aggregation Over Bursty Event Streams. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 1452–1464. https://doi.org/10.1145/3448016.3452785

[37] Gururaghav Raman, Jimmy Chih-Hsien Peng, Bo Zhao, and Matthias Weidlich. 2019. Dynamic Decision Making for Demand Response through Adaptive Event Stream Monitoring. In *2019 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 1–5.

[38] Medhabi Ray, Chuan Lei, and Elke A. Rundensteiner. 2016. Scalable Pattern Sharing on Event Streams. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 495–510. https://doi.org/10.1145/2882903.2882947

[39] Wolf Rödiger, Sam Idicula, Alfons Kemper, and Thomas Neumann. 2016. Flow-join: Adaptive skew handling for distributed joins over high-speed networks. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 1194–1205.

[40] Lukas Rupprecht, William Culhane, and Peter Pietzuch. 2017. SquirrelJoin: Network-aware distributed join processing with lazy partitioning. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1250–1261.

[41] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter R. Pietzuch. 2009. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS 2009, Nashville, Tennessee, USA, July 6-9, 2009*, Aniruddha S. Gokhale and Douglas C. Schmidt (Eds.). ACM. https://doi.org/10.1145/1619258.1619264

[42] Utkarsh Srivastava, Kamesh Munagala, and Jennifer Widom. 2005. Operator placement for in-network stream query processing. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART sym posium on Principles of database systems*. ACM, 250–258.

[43] Fabrice Starks, Vera Goebel, Stein Kristiansen, and Thomas Plagemann. 2018. Mobile Distributed Complex Event Processing - Ubi Sumus? Quo Vadimus? In *Mobile Big Data, A Roadmap from Models to Technologies*, Georgios Skourletopoulos, George Mastorakis, Constandinos X. Mavromoustakis, Ciprian Dobre, and Evangelos Pallis (Eds.). Lecture Notes on Data Engineering and Communications Technologies, Vol. 10. Springer, 147–180. https://doi.org/10.1007/978-3-319-67925-9_7

[44] Fabrice Starks and Thomas Peter Plagemann. 2015. Operator placement for efficient distributed complex event processing in MANETs. In *11th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2015, Abu Dhabi, United Arab Emirates, October 19-21, 2015*. IEEE Computer Society, 83–90. https://doi.org/10.1109/WiMOB.2015.7347944

[45] Kia Teymourian, Malte Rohde, and Adrian Paschke. 2012. Knowledge-based processing of complex stock market events. In *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, Elke A. Rundensteiner, Volker Markl, Ioana Manolescu, Sihem Amer-Yahia, Felix Naumann, and Ismail Ari (Eds.). ACM, 594–597. https://doi.org/10.1145/2247596.2247674

[46] David Waitzman, Craig Partridge, and Stephen E Deering. 1988. *Distance vector multicast routing protocol*. Technical Report.

[47] John Wilkes. 2020. Yet more Google compute cluster trace data. Google research blog. Posted at https://ai.googleblog.com/2020/04/yet-more-google-compute-cluster-trace.html..

[48] Bo Zhao, Nguyen Quoc Viet Hung, and Matthias Weidlich. 2020. Load Shedding for Complex Event Processing: Input-based and State-based Techniques. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 1093–1104. https://doi.org/10.1109/ICDE48307.2020.00099

[49] Bo Zhao, Han van der Aa, Thanh Tam Nguyen, Quoc Viet Hung Nguyen, and Matthias Weidlich. 2021. EIRES: Efficient Integration of Remote Data in Event Stream Processing. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2128–2141. https://doi.org/10.1145/3448016.3457304

[50] Yongluan Zhou, Ying Yan, Feng Yu, and Aoying Zhou. 2006. Pmjoin: Optimizing distributed multi-way stream joins by stream partitioning. In *International Conference on Database Systems for Advanced Applications*. Springer, 325–341.