Network-Aware Stream Query Processing in Mobile Ad-Hoc Networks

Dan O'Keeffe
Imperial College London

Theodoros Salonidis
IBM T.J. Watson Research Center

Peter Pietzuch
Imperial College London

***Abstract*—Many real-time decision support and sensing applications can be expressed as continuous stream queries over time-varying data streams, following a data stream management model. We consider the problem of the efficient and resilient execution of continuous stream queries in tactical edge networks formed from mobile ad-hoc networks (MANETs) with limited backend connectivity. Previous approaches for distributed stream query execution target data center environments in which networks are static, and centralized control is feasible. The distributed, bandwidth-constrained and highly dynamic nature of MANETs render such approaches insufficient—while a stream query executes in a MANET, changes in the network topology mean that any fixed query plan eventually becomes outdated.**

We introduce an adaptive, *network-aware* approach for stream query planning in MANETs, which supports both single- and multi-input windowed stream query operators. The basic idea is to increase the path diversity available when executing stream queries by replicating query operators across many nodes in the MANET. During execution, it becomes possible to dynamically switch between different operator replicas based on connectivity and other network path conditions. We evaluate our approach in emulated MANETs, showing that it can increase substantially the robustness of distributed stream query processing under mobility.

I. Introduction

A *data stream management* (DSM) model [4] naturally expresses the behavior of many real-time decision support, sensing and analytics applications. In contrast with a traditional database model, which executes one-shot queries on stored data, a DSM model executes *continuous* queries on real-time streaming data. A continuous *stream* of data items is processed by a query that extracts knowledge in real time. For example, troops at a spot checkpoint with cameras or head-mounted displays can execute a stream query to continuously detect, recognize and share any suspicious enemy faces, vehicles or motion within 10 m of the checkpoint (see Figure 1).

To implement a DSM model, a data stream management system (DSMS) typically represents queries as *dataflow graphs*. In such a dataflow graph, vertices correspond to stream query operators and directed edges describe the movement of data items between operators [4], which can be database tuples, network measurements, location readings, or images from video sources. The operators are executed by the DSMS and can perform simple arithmetic operations, data filtering or aggregation and more computationally-intensive database, signal processing or data mining operations on the stream data.

DSMSs have been used to realize many commercial applications, including click-stream analysis, network intrusion detection, video stream analysis and financial risk calculations.

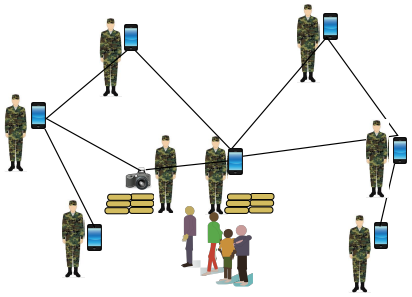


Fig. 1: Checkpoint scenario. Soldiers equipped with smartphones communicate over a MANET

Such applications typically run in centralized data centers with high volumes of data traffic that are processed by high-performance networked compute clusters.

In contrast, mobile devices in tactical environments are resource- and energy-constrained and have intermittent or bandwidth-limited network connectivity to backend services. Sources of stream data and processing resources are distributed, and connected via mobile ad-hoc networks (MANETs). An individual device such as a smartphone or a tablet may not have sufficient compute resources to carry out all required processing by itself, which requires a *distributed* processing model that combines the resources of multiple devices. A vital requirement is the *resilience* of critical mission operations, and the dynamic nature of MANETs makes it more challenging to meet this requirement for distributed DSM applications.

Our key observation is that, for a DSM model to become feasible in a MANET, the execution of stream queries must adapt to network conditions, thus making it *network-aware*. In this paper, we propose a novel approach for *adaptive stream query planning* that takes the current conditions of the MANET into account in order to increase data processing resilience and throughput. While adaptive query planning has been explored by the database research community to cope with changes in workloads in a datacenter environment [8], our goal is to perform query planning in a manner that reacts to changes in the network conditions in MANETs.

Our network-aware approach to adaptive query planning consists of two parts: (i) operators in the dataflow graph are *replicated* within the MANET in a manner that provides robust availability and access in the face of intermittent network connectivity and congestion; and (ii) data stream items are then *routed* to operator replicas over the highest quality available

network paths. Our experiments show that the additional *path diversity* operator replication affords results in more efficient and robust stream processing compared to using a single static deployment plan for a query.

In summary, the paper makes the following contributions:

- a novel *replicated dataflow graph model* for distributed data stream processing in MANETs, under which a number of replicated query operators are deployed to a subset of nodes in the network to support network dynamicity;
- a network-aware cost-based routing algorithm that dynamically sends data to different operator replicas in order to maximize query performance. The algorithm ensures consistent switching decisions for both single and multi-input query operators; and
- an evaluation of the performance benefits of operator replication for a distributed face recognition application.

Next we provide background on DSM (Section II) and introduce our adaptive network-aware approach to query planning in MANETs (Section III). We then discuss the implementation of a prototype DSMS for executing streaming queries that realizes our approach (Section IV), and evaluate our prototype's performance with an emulated MANET (Section V). We finish with related work (Section VI) and conclusions (Section VII).

II. Background

A. Data stream management model

We adopt the following formal DSM model in this paper:

Data model. A stream s is an infinite series of tuples $t \in s$. A *tuple* $t = (\tau, p)$ has a logical timestamp τ and a payload p. The timestamp $\tau \in \mathbb{N}^+$ is assigned by a monotonically increasing logical clock of an operator. Tuples in a stream are ordered according to their timestamps.

Operator model. Tuples are processed by operators. A logical *operator* o takes n *input streams*, denoted by the set $I_o = \{s_1, \ldots, s_n\}$, processes their tuples and produces a logical *output stream* O_o. The notation $I_o[\bar{\tau}]$ specifies all tuples in the input streams with timestamps less than $\bar{\tau}$ where $\bar{\tau} = (\tau_1, \ldots, \tau_n)$. We assume that all operators are *windowed* operators in that they divide their input streams into finite windows of tuples before processing them. Windows may be *count-* or *time-based*, modeled using *size* and *slide* parameters [2].

Query model. A query is specified as a *dataflow graph* $Q = (\mathbb{O}, \mathbb{S})$ where $\mathbb{O}$ is the set of operators and $\mathbb{S}$ is the set of streams. A stream $s \in \mathbb{S}$ is a directed edge between two operators, $s = (o, o')$, where $\{o, o'\} \subseteq \mathbb{O}$. A query has two special operators, *src* and *sink*, that act as the sources and sink for data streams, respectively. An operator u is *upstream* to o, denoted by $u \in up(o)$, when $\exists (u, o) \in \mathbb{S}$. Similarly, an operator d is *downstream* to o, $d \in down(o)$, when $\exists (o, d) \in \mathbb{S}$. Finally, to optimize query execution, a DSMS first transforms Q into a *query plan* $G = (\mathbb{O}, \mathbb{S})$, also a dataflow graph.

B. Problem statement

The problem that we address is how to design a DSMS that supports the above DSM model and can operate in a tactical environment with a MANET. We assume that in this environment (i) applications are too compute-intensive to execute on a single node, and (ii) distributed computation is challenging to coordinate due to the dynamic topology of a MANET. An additional constraint is that the DSMS should not unnecessarily waste scarce resources such as network bandwidth or node energy, and should allow for correlation of stream data across multiple sources.

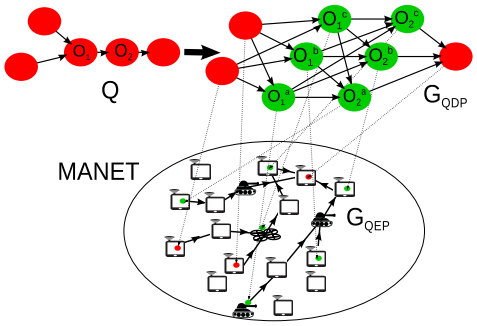


Fig. 2: Operator replication and query execution in a MANET

We assume that, although some or all network nodes may be mobile and the network can become partitioned due to mobility, most of the time there will exist a path in the network from sources to sinks. We argue that scenarios in which nodes are nearly always disconnected are better suited to store-and-forward models such as *delay-tolerant networking* [9].

In addition, we assume that for most distributed DSM applications executing in a MANET, the network will be the bottleneck due to its dynamic nature and the reduced bandwidth and potential for interference in wireless networks. Cellular or satellite connectivity is unavailable, intermittent, or too bandwidth-constrained to make offloading computation to a centralized backend a feasible solution.

III. Adaptive Network-Aware Stream Queries

We describe a new approach for executing data stream queries in MANETs that relies on network-aware adaptation of the query plan that the DSMS executes. The plan adaptation has two steps: (i) for a given query, the DSMS generates a query plan with multiple *replicated* operators; and (ii) when executing the query, the DSMS routes data tuples along *redundant* processing paths in the query plan depending on the current network conditions in the MANET.

A. Query deployment

Given a query Q, a query planner that is part of the DSMS first decides on a physical query *deployment plan* G_{QDP}. As shown in Figure 2, the query deployment plan is a dataflow graph that, in contrast to the logical query graph Q, contains *replicated operators*. Operator replication gives the DSMS a choice at runtime to which downstream replica a given operator sends its stream output for further processing. This enables the DSMS to avoid network bottlenecks and thus improve query throughput (see Section III-B).

The query planner precomputes the number of replicas for each operator based on the expected characteristics of the network, such as the expected level of mobility. We assume that only approximate aggregate information about the network

is available to the query planner at deployment time, such as the expected network size and density, and the average level of mobility. The planner then deploys each operator replica in G_{QDP} to a node in the MANET. We treat the problem of replica placement in the network as an orthogonal issue; for simplicity, we assume simple random placement.

B. Query execution planning

Having deployed a physical query deployment plan G_{QDP}, the challenge for the DSMS is to determine the best physical *execution plan* for the query. A query execution plan G_{QEP} is a subgraph of G_{QDP} that defines for each operator replica in G_{QDP} the downstream replica that it should send its output tuples to. Figure 2 depicts the relationship between the deployment plan graph G_{QDP} and the current G_{QEP} for a query Q. Each link in G_{QEP} corresponds to a multi-hop network-level path in the MANET with an associated cost.

Our goal is to make execution planning decisions in a *network-aware* manner. Due to the dynamic nature of MANETs, the cost of communication between nodes can change dramatically over time. DSMS execution planning must therefore continuously monitor the network and *adapt* G_{QEP} in response to cost changes. Furthermore, execution planning should be *distributed* to avoid relying on a centralized coordinator as part of the DSMS because node failures and disconnections are likely in a MANET.

Dynamic routing. We propose a decentralized algorithm for query execution planning in a DSMS that views the replica selection problem as a distributed *routing* problem. Periodically, operator replicas measure the network costs to their direct downstream replicas in G_{QDP}. Costs can be provided by the underlying MANET protocol or measured directly through explicit probing. Replicas then communicate to update the costs associated with their local copies of G_{QDP}. Finally, each replica independently determines its best downstream replica by computing the shortest path to the sink operator over G_{QDP}, and selects the next-hop downstream replica on this path.

This distributed dynamic shortest-path routing algorithm has two key benefits: (i) intermediate operators can change the path of tuples in-flight in response to changing network conditions, allowing more flexibility than e.g. strict source-routing; and (ii) since routing decisions are based on the cost to the destination, this approach avoids sending tuples along a path with low cost to the next downstream replica but high cost to the destination, as may occur with greedy local routing.

The DSMS uses additive costs to determine the overall network cost of the query. This enables it to use shortest-path algorithms of low complexity such as Dijkstra or Bellman Ford. In our DSMS implementation, we use a modified version of Dijkstra's shortest path algorithm.

Tuple routing algorithm. Algorithm 1 describes the routing behavior of each operator replica in more detail. Given an operator replica r, we define its *replica* deployment plan graph G^r_{QDP} as the subgraph of the global deployment plan graph G_{QDP} containing all transitively reachable downstream replicas. Periodically, each operator executes the *adapt-route* function to determine whether it needs to switch to a different downstream replica (line 5). The function first computes the shortest path to the sink using the operator's current replica deployment plan graph (line 6), and then compares the difference in cost between the new shortest path and the currently active path (line 7). Finally, the function returns the new path if it offers sufficient benefit (line 8), and otherwise returns the currently active path (line 10).

Algorithm 1 Dynamic tuple routing algorithm

```
1:  id ∈ ID // local node id
2:  sink ∈ ID // sink id
3:  G^id_qdp ⊂ G_qdp // Deployment plan graph for replica id
4:  P_curr // The currently active path to the sink from this node
5:  function ADAPT-ROUTE( id, sink, G^id_qdp, P_curr )
6:      P_new = SHORTEST-PATH(id, sink, G^id_qdp)
7:      if COST(P_curr, G^id_qdp) − COST(P_new, G^id_qdp) > 0 then
8:          return P_new
9:      else
10:         return P_curr
```

Support for multi-input operators. Although the above algorithm computes an efficient execution plan for simple chain queries, it does not work for more complex queries with multi-input operators such as stream joins [2]. Multi-input operators need tuples from both input streams within a window in order to produce a correct result. If two input sources route their tuples independently to two different replicas, the result will be incorrect—instead, both inputs must *coordinate* their routing decisions.

To solve this problem, we introduce a technique called *multi-input cost aggregation*. Upstream operators measure the costs to their downstream multi-input replicas as before, but instead of using those costs directly, they send them downstream to the corresponding multi-input replicas. The replicas compute the *aggregate* cost across all of their direct upstreams, and return this cost. The upstream operators then combine the aggregate costs with other G_{QDP} updates received from their downstreams to construct their local copy of G_{QDP}. Finally, each operator replica computes the shortest path as before.

Multi-input cost aggregation ensures that the aggregate cost to each multi-input replica is the same for each input. Since the lowest cost path from a multi-input replica to the sink is independent of the inputs, each operator making a routing decision will compute the same shortest path to the destination, thus implicitly coordinating their choice of downstream.

IV. Implementation

We next describe the implementation of a prototype DSMS that incorporates our adaptive query planning technique.

A. System architecture

Our prototype is based on SEEP [10], a Java-based DSMS designed originally for datacenters. A SEEP cluster consists of a master node and one or more worker nodes. Workers join the cluster by registering with the master. Users submit new queries to SEEP through the master, which then deploys query operators to the workers for execution. Unlike SEEP, our prototype workers run on MANET nodes, and the master runs on either a MANET node or on a node outside the MANET if

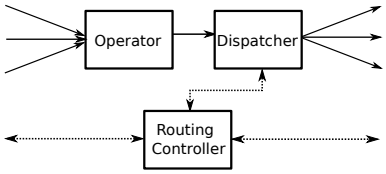


Fig. 3: Node worker architecture in the DSMS prototype

available. The master is only used for initial query deployment, and could be decentralized with additional engineering effort.

Query deployment. For the initial conversion of stream queries into a physical dataflow graph with several replicas for each operator, we implement a query planner component as part of the SEEP master. The planner takes as input a stream query Q, creates G_{QDP} from Q and then deploys each operator replica in G_{QDP} to a random worker node in the MANET. The planner implementation extends SEEP's scale-out capability, designed for replication of partitionable data-parallel operators, to support replication of arbitrary operators.

Runtime query execution. A SEEP worker is responsible for receiving and processing input tuples, and sending any operator output tuples to downstream worker nodes. Workers implement the routing functionality needed to implement our network-aware routing algorithm (Section III-B).

Figure 3 shows the high-level architecture of each DSMS worker. The operator component receives tuples from its upstream operators and outputs them to a *dispatcher*. The dispatcher forwards output tuples to a downstream replica. The dispatcher consults the *router* to determine the replica to select. The router of replica r selects a downstream replica based on its view of the dataflow graph G^r_{QDP}, over which it periodically computes the lowest cost path to the sink.

The router is responsible for maintaining the costs associated with each link in G^r_{QDP}. In the simplest case, the network layer uses a link-state routing algorithm and allows the application to access its routing tables. The router consults the network layer periodically to obtain G_{net}, the current network-level link-state. The router then computes the cost for each application-level link (o, o') in G^r_{QDP} as the cost of the shortest path between o and o' in G_{net}.

If the network-layer does not use a link-state routing algorithm or does not allow the application layer to query its routing tables, each replica r can measure itself only the network-level costs to its direct downstreams in G^r_{QDP}. In such cases, the router of r periodically obtains the costs for the remaining links in G^r_{QDP} from the downstreams of r. In turn, the router of r periodically forwards G^r_{QDP} to r's upstreams.

B. Network-layer interaction

Routing. Our implementation is agnostic to the network-layer routing algorithm used in the underlying MANET, although the algorithm chosen may impact the performance of query processing. Our evaluation (Section V) uses OLSR [6], a *proactive* routing protocol. *Reactive* routing protocols (e.g. AODV [15]), in which nodes gather routing information on-demand in response to send requests, are also applicable.

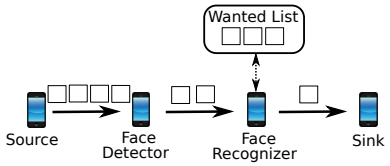


Fig. 4: Stream query for face recognition application

Cost metrics. Our implementation can use any additive cost metric for routing, including hop count, latency, packet loss, or ETX [7]. We require additive costs to support shortest-path routing algorithms of low complexity (e.g. Dijkstra). Costs can be supplied by the network layer, or if necessary measured at the application layer using explicit probing (e.g. bandwidth measurements). Our evaluation (Section V) uses ETX, a high-performance additive metric for wireless mesh networks, defined as the inverse of packet success ratio. Our architecture also allows us to incorporate node costs such as processing load or energy remaining.

C. Mobility models

We do not assume a particular mobility model, allowing for example a query deployment phase that randomly deploys replicas when accurate location predictions are hard to make a priori (e.g. for a random waypoint mobility model). More sophisticated mobility models could involve nodes that are known to follow certain paths or congregate in specific areas, or incorporate information about group or social relationships. Our query planning framework could use such information to influence both the level of operator replication and operator placement. We leave this for future work.

V. Evaluation

We evaluate the performance of our adaptive network-aware query planning in an emulated mobile ad-hoc network, focusing on the distributed face recognition application as shown in Figure 1, and also in simulation. Our experiments explore the effect of operator replication on processing throughput with different levels of network dynamicity, density and size.

A. Emulation experiments

Distributed face recognition query. The query for our distributed face recognition application is shown in Figure 4. A source captures frames from a local video feed and forwards them to a (1) *face detection operator*. It analyses each frame and forwards the image coordinates of detected faces to a (2) *face recognition operator*, which compares input faces to a local database of persons of interest. The faces and names of matches are sent to a sink, which displays the results to the user. In our experiments, all operators have a selectivity of 1 because the goal is to make a continuous video stream available at the sink, with throughput measurements indicating the sustained transmission rate.

Experimental setup. We deploy the query in a MANET that is emulated using the CORE/EMANE network emulator (version 0.9.1) [1]. Nodes in the MANET move according to a *random waypoint* mobility model with a pause time of 2 ± 1 seconds (node speed varies according to the experiment). At the physical layer, we set the node transmit power

to −10.0 dBm and the path loss mode to 2ray, giving an approximate transmission range of 500 m with stable TCP throughput. At the MAC layer, we use 802.11b with a unicast and multicast rate of 11 Mb/s. We use the default values for all other EMANE parameters. At the network layer, we use OLSR [6] with ETX [7] as a cost metric.

1) Impact of network mobility: Our first experiment evaluates the benefit of operator replication in a 25 node network with different levels of node mobility. Figures 5 and 6 show the query throughput and latency with different replication factors k, as we increase node speed (error bars indicate the standard deviation across 5 runs).

The benefits of replication for throughput become apparent as node speed increases, even though the absolute throughput falls. With an average node speed of 5 m/s, there is a 1.6× and 1.8× improvement in mean throughput for $k = 2$ and $k = 3$, respectively. At 10 m/s, the improvement increases to 2.1× ($k = 2$) and 2.6× ($k = 3$). Even with low mobility (1 m/s), a replication factor of $k = 2$ gives a mean speedup of around 1.3×, and $k = 3$ results in 1.4×.

The variance is slightly higher with low mobility because the initial operator and node placement has more influence on the throughput. Even though the variance is higher, the relative throughput increases with larger replication factors.

Figure 6 shows that query latency remains low for all values of k up to 10 m/s, allowing for near real-time distributed processing. However, variance in latency increases considerably for higher levels of mobility. In absolute terms, the mean 95$^{\text{th}}$ percentile latency is sub-second with replication for low mobility (1 and 5 m/s), and on the order of a second at 10 m/s.

In terms of relative latency, both $k = 2$ and $k = 3$ give a 50% reduction in latency over $k = 1$ at 1 m/s, and 50% and 40%, respectively, at 5 m/s. At 10 m/s, latency for $k = 2$ is on par with $k = 1$, and with $k = 3$ it is approximately 20% lower.

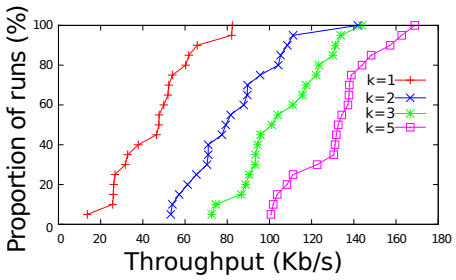
2) Scalability: The next experiment explores how increasing the network size, while keeping network density fixed, affects throughput for different replication factors k. Figure 7 shows the throughput for different replication factors in a network with 50 nodes and an average node speed of 5 m/s.

Although the absolute throughput is lower than for the 25-node network in Figure 5, replication still improves performance considerably. For $k = 2$, there is a 1.8× speedup in mean throughput, and a 2.3× improvement for $k = 3$. We consider a 50-node network as an upper bound for the size of a realistic deployment, and thus the results show the applicability of our approach even at scale.

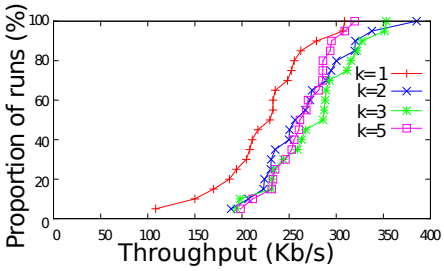
B. Simulation experiments

To characterize the behaviour of our approach for a wider variety of environmental parameters, we conduct several additional experiments in simulation using the JIST/SWANS wireless ad-hoc network simulator [5].

Experimental setup. All of our simulations use a 25-node network. The effective transmission range for each node is 625 m. Since JIST does not provide an OLSR implementation, our simulator experiments use the AODV routing algorithm, and the hop count provided by AODV as a cost metric.



(a) Sparse network



(b) Dense network

Fig. 8: Throughput percentiles in dense (1500 m ×1500 m) and sparse (3135 m ×3135 m) networks for a query with 3 operators, avg. node speed 4 m/s, and replication k

1) Impact of network density: Our first set of simulator experiments investigates the effect of reduced network density on the relative throughput achieved for different replication factors k. Figure 8 compares the throughput with an average node speed of 4 m/s in (a) sparse and (b) dense networks.

In a dense network, $k = 2$ is beneficial, $k = 3$ gives diminishing returns, and increasing k further reduces throughput. For sparse networks, $k > 3$ still gives a clear benefit in throughput, although the absolute throughput is lower than for the dense case. These differences in behaviour arise because in a dense network, the worst case path length is shorter, and so, on average, the gain from replica switching is lower. Since higher replication factors have more chances to switch, switching overhead begins to outweigh the benefits of replication.

2) Switching threshold: Intuitively, the benefit of replication factors $k > 3$ should further diminish in a dense network as dynamicity increases, because new paths retain their advantage for shorter periods. To confirm this intuition, Figure 9 repeats the previous experiment with increased node mobility (10 m/s) and different *switching thresholds*. Instead of always switching to the lowest cost path every period, nodes only switch if the difference between the expected cost of the new path and the expected cost of the current path is above a certain threshold.

In Figure 9, we measure the throughput for different replication factors in a dense network with high mobility and switching thresholds of (a) 1 hop and (b) 3 network hops. The results show that an increased switching threshold improves throughput for higher replication factors by avoiding unnecessary switches.

VI. Related Work

Much research work in the database community exists on *adaptive query processing* [8]. Unlike traditional "optimize-then-execute" query processing, the goal of adaptive planning

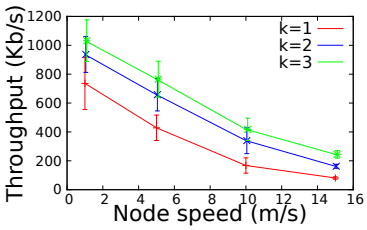


Fig. 5: Face recognition. Throughput vs. mobility for replication k.

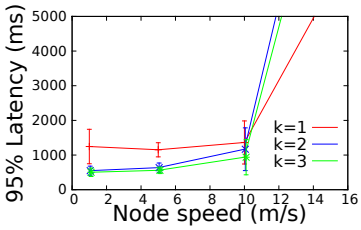


Fig. 6: Face recognition. Latency vs. mobility for replication k.

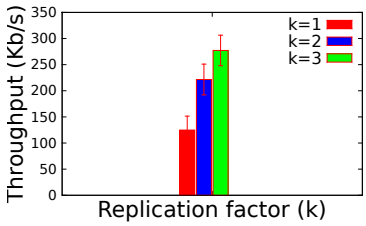
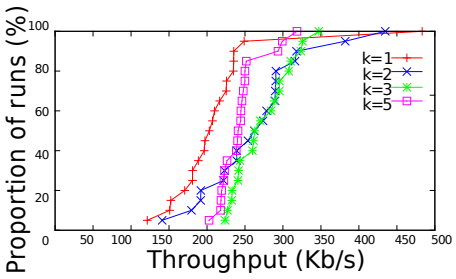
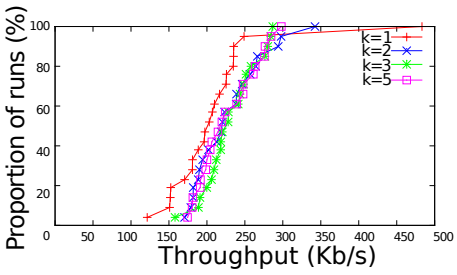


Fig. 7: Face recognition. Throughput in a large network with replication k.



(a) Switch threshold = 1 hops



(b) Switch threshold = 3 hops

Fig. 9: Throughput percentiles with different switching thresholds for a 3-operator query with replication k in a 1500 m $\times$1500 m area with avg. node speed 10 m/s

is to use runtime feedback to modify query processing dynamically to improve response times or utilization. Rundensteiner et al. give an overview of the design space [14], ranging precomputed static plans to dynamic per-tuple routing.

A plethora of work exists on static query plan generation for DSM systems. Query planners such as SODA [16] and SQPR [13] formulate query planning and placement as an optimization problem and solve it using standard techniques like mixed integer linear programming (MILP) [12]. We can exploit these approaches to generate an initial query deployment plan from a given logical dataflow graph.

Adaptive execution for streaming queries has been considered in Eddies [3] and QueryMesh [14]. These approaches adapt to changes in input data by either selecting one of multiple pre-computed query plans [14], or adapting the order of operators visited by tuples while ensuring a correct query result [3]. Both approaches are typically executed in single-node or static network environments and therefore do not apply for dynamic network environments such as MANETs. They also focus on relational database queries while we consider queries with more general operators (e.g. face recognition).

In the context of DSM systems, Hwang et al. describe how replicated plans with multiple paths between a tuple source and sink and redundant data processing can improve reliability or reduce processing latency [11]. Instead of sending a copy of each tuple along every path, we send only a single tuple, and paths along which successive tuples travel may differ. While replication-based approaches are likely to perform better in extremely dynamic networks, they incur the cost of redundant processing, increased network traffic and interference.

VII. Conclusion

In this paper, we proposed the data stream management (DSM) model as a suitable foundation for real-time decision support and analytics applications in tactical environments, where nodes may have limited backend connectivity. However, existing approaches for DSM that target data centers are unsuited to MANETs—the dynamic nature of the network means that any fixed query deployment plan quickly becomes outdated, resulting in poor performance. We describe instead an adaptive *network-aware* approach to stream query planning where (i) operators are *replicated* within the MANET and (ii) data streams are *routed* to operator replicas over the best available network path. We showed that the additional path diversity afforded by operator replication results in more efficient and robust stream processing in MANETs.

References

[1] J. Ahrenholz. Comparison of CORE network emulation platforms. In *MILCOM '10*.

[2] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal*, 2006.

[3] R. Avnur and J.M. Hellerstein. Eddies: Continuously adaptive query processing. In *ACM SIGMOD Record*, 2000.

[4] H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, et al. Retrospective on Aurora. *VLDB Journal*, 13(4), 2004.

[5] R. Barr, Z.J. Haas, and R. van Renesse. JiST: An efficient approach to simulation using virtual machines. *Software Practice and Experience*, 2005.

[6] T. Clausen and P. Jacquet. Optimized link state routing protocol (OLSR), 2003.

[7] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MOBICOM '03*.

[8] A. Deshpande, Z. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 2006.

[9] K. Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03*.

[10] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch. Integrating Scale Out and Fault Tolerance in Stream Processing using Operator State Management. In *SIGMOD '13*.

[11] J.-H. Hwang, U. Çetintemel, and S. Zdonik. Fast and reliable stream processing over wide area networks. In *ICDEW '07*.

[12] IBM. ILOG CPLEX. www.ibm.com, 2010.

[13] E. Kalyvianaki, W. Wiesemann, Q. Hieu Vu, D. Kuhn, and P. Pietzuch. SQPR: Stream query planning with reuse. In *ICDE*, 2011.

[14] R.V. Nehme, K. Works, C. Lei, E.A. Rundensteiner, and E. Bertino. Multi-route query processing and optimization. *Journal of Computer and System Sciences*, 2013.

[15] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing, 2003.

[16] J. Wolf, N. Bansal, K. Hildrum, S. Parekh, et al. SODA: An optimizing scheduler for large-scale stream-based distributed computer systems. In *Middleware*, 2008.