Comet: Decentralized Complex Event Detection in Mobile Delay Tolerant Networks

Jianxia Chen, Lakshmish Ramaswamy
Department of Computer Science
The University of Georgia
Athens, GA 30602
{chen, laks}@cs.uga.edu

David K. Lowenthal
Department of Computer Science
The University of Arizona
Tucson, AZ 85721
dkl@cs.arizona.edu

Shivkumar Kalyanaraman
IBM Research India
Bangalore 560071 India
shivkumar-k@in.ibm.com

***Abstract*—Increased commodity use of mobile devices with sensory capabilities has the potential to enable mission-critical monitoring applications in various domains. However, these mobile-enabled monitoring applications have to often work in environments where a delay-tolerant network (DTN) is the only feasible communication paradigm. DTNs are multi-hop networks prone to long delays and frequent disruptions. Unfortunately, there is a lack of effective and scalable support for building sophisticated DTN-based monitoring applications. Detection of complex (composite) events, for example, is fundamental to monitoring applications. However, there is little work in designing scalable complex event detection (CED) techniques for DTNs. The current plan-based CED techniques are mostly centralized, and hence are inherently unscalable for DTNs.**

In this paper, we create *Comet*, – a decentralized plan-based, efficient and scalable CED for DTNs. Comet shares the task of detecting complex events (CEs) among multiple nodes, with each node detecting a part of the CE by aggregating two or more primitive events or sub-CEs. Comet uses a unique *h-function* to construct cost and delay efficient *CED trees*. As finding an optimal CED *plan* requires exponential-time, Comet finds near-optimal detection plans for individual CEs through two novel heuristic planning techniques: *multi-level push-pull conversion* and *virtual CED tree creation*. Performance results show that Comet reduces cost by up to 89% compared to pushing all primitive events and over 60% compared to a two-level exhaustive search algorithm.

I. Introduction

Recent years have seen the advent and large-scale proliferation of mobile devices such as smart phones and different types of mobile wireless sensors that are capable of acquiring and disseminating various kinds of sensory data. This has the potential to enable sophisticated monitoring applications. However, these devices are increasingly being used in environments where infrastructure limitations preclude continuous, reliable end-to-end network connectivity. Examples of such domains include battlefield surveillance and monitoring of remote rural areas, seismology, interplanetary space exploration, and oceanography.

Delay-tolerant networking (DTN) has been proposed and is being increasingly adopted as the networking architecture for such environments [1], [2], [3]. DTNs exist outside of the Internet; DTN links are characterized by long delays, frequent interruptions, and high error rates [4]. Research on DTNs has primarily focused on routing issues either for unicasting or for multicasting [5], [3], [6], [7], [8]. Only recently have researchers started to study issues such as caching for DTNs [9].

Unfortunately, despite the apparent importance of monitoring applications for these domains, there is little research on providing effective and efficient support for building them. Several of the essential components of modern monitoring applications have been designed for Internet-based environments where the underlying network provides continuous, reliable, low-latency end-to-end connectivity, and they do not work well in a mobile-DTN setting. A prime example is the *complex event detection (CED)* component. *Complex events (CEs)* are composed (using various operators) from multiple atomic, possibly geographically distributed *primitive events* (PEs) [10], [11], [12], [13].

Most of the popular CED techniques are based on plans that rely, explicitly or implicitly, on centralized processing of PEs. However, centralization is unacceptable in networks that are prone to long delays and frequent disruptions. While the cost and latency imposed by a centralized CED framework may be tolerable for high-bandwidth wired networks, it is prohibitively expensive for DTNs. There are three reasons for this. First, centralization prevents exploiting topological proximity of event sources. Generally, communication overhead is lower when PEs are processed from nearby sources, and then partial CEs are detected and forwarded to the sink. This is because nearby processing allows early elimination of PEs that cannot be part of a CE—as opposed to sending all PEs to the sink directly. Second, most existing CED techniques assume network connectivity between various PE sources and the event destinations almost always exists; this does not hold for DTNs, so PEs cannot be pulled at arbitrary times. Moreover, pushing of some PEs is necessary because of rare DTN availability. This assumption does not hold for DTNs where certain links might be unavailable for long durations. Third, DTNs operate on a store-and-forward paradigm, and hence PEs will be stored at intermediate nodes until the next link becomes available. It is advantageous to process PEs and detect sub-CEs at intermediate nodes when waiting for the availability of the next link. While there are a few decentralized CED systems [14], [15], they are not plan-based. Planning is important for comprehensively exploring the cost-delay tradeoffs so that the system utilizes the available

resources in an optimal fashion.

This paper contributes a novel, decentralized multi-stage framework, called *Comet*, for efficient and scalable CED in mobile-DTNs. To the best of our knowledge, Comet is the first plan-based decentralized system. Comet supports distribution of the CED process among multiple nodes, with each node detecting a part of the CE (sub-CE) by aggregating two or more PEs or sub-CEs. Because finding an optimal (lowest) cost plan is NP-complete, Comet includes a heuristic planning algorithm that, given a CED delay tolerance, derives a CED plan with low cumulative communication cost. The CED plan output by Comet specifies where (on which nodes) the sub-CED tasks are hosted, which constituent events are pushed to that node, which constituent events are pulled by that node, and in what order. Comet can exploit both single target pulls (in which instances from a single event source are retrieved) and multi-target pulls (in which event instances from multiple sources are retrieved), and it incorporates several novel features.

- First, we present a technique to construct a cost and delay efficient detection tree for a CE given the destination and the PE sources. Ideally, the CED tree should minimize both the cost and the delay of transferring PE instances from the source to destination. However, it is often impossible to achieve both. We combine these two factors into a novel *h-function* and use it to guide our tree construction.
- We design a multi-level push-pull conversion mechanism that can work in conjunction with multi-level CED trees. Our cost-delay sensitive heuristic algorithm operates in two phases: first by converting as many proactive push operations as possible into single-target pulls; and then by converting as many of the remaining pushes as possible into multi-target pulls.
- Third, any planning strategy that works at the granularity of links suffers from the limitation that it cannot explore certain plans. Specifically, suboptimality can result from the need for certain PEs to be pushed along a particular link while other PEs are pulled along the same link. Comet overcomes this limitation by creating multiple virtual topologies through a unique technique called *shorting*.

We have run extensive experiments with Comet. Performance results show that Comet reduces cost in some topologies by over 89% compared to pushing all primitive events, and over 60% compared to a two-level exhaustive search algorithm. Moreover, in most topologies, Comet outperforms all other techniques, often by similar margins. This includes both skewed topologies and topologies with increasing depth.

II. Background and Challenges

In this section we briefly discuss the fundamentals of CED and DTN. Then, we formally state the problem of CED on DTNs and explain why the existing CED techniques are not appropriate for DTNs.

A. DTN

DTNs are mobile networks in which continuous, bi-directional, end-to-end connectivity between two arbitrary hosts is not guaranteed. The links (also referred to as *contacts*) of a DTN are characterized by intermittent connectivity (depending upon when the end-nodes of a link come within each other's wireless range), asymmetric data rates, and high error rates. DTNs operate on a store-and-forward paradigm where a node stores a packet it receives until an onward link towards its destination becomes operational. Based on the temporal link connectivity characteristics, DTNs can be classified into two broad classes: *scheduled-contacts DTNs* and *opportunistic-contacts DTNs*. In scheduled-contacts DTN, the contacts among nodes occur according to a schedule, as opposed to in an ad-hoc manner in opportunistic-contacts DTNs. In other words, the up and down times of the links of a scheduled contacts DTN can be predicted to a reasonable degree of accuracy. The data ferry network that uses buses/trams operating according to a specified schedule is an example of scheduled-contact DTN. In this paper, we confine our discussion to scheduled-contacts DTNs.

Our discussion is based on the following conceptual model of scheduled-contact DTNs. The DTN is composed of N nodes represented as $\{V_1, V_2, \ldots, V_N\}$. A link is the intermittent connection between two nodes. The link between nodes V_f and V_g is represented as L_{fg}. Each link is associated with five properties. The expected disconnection period of the link L_{fg}, represented as $EDP(L_{fg})$, is the time duration between two consecutive active sessions of the link. In other words, once L_{fg} becomes disconnected, it is expected to remain dormant for $EDP(L_{fg})$. Analogously, the expected active period of L_{fg}, represented as $EAP(L_{fg})$, is the time duration for which L_{fg} is expected to remain active after gaining connectivity. The bandwidth of L_{fg}, denoted $BW(L_{fg})$, is the number of bytes per second that can be transferred over L_{fg} when the link is active. The latency of L_{fg} (represented as $LT(L_{fg})$) is the time required for a packet to travel from V_f to V_g when the L_{fg} is operational. Generally, $EDP(L_{fg}) \gg LT(L_{fg})$. The worst case delay in transferring a packet along L_{fg} is denoted as $DL(L_{fg})$. Although $DL(L_{fg})$ is equal to the sum of $EDP(L_{fg})$ and $LT(L_{fg})$, it can be approximated as $EDP(L_{fg})$ (since $EDP(L_{fg}) \gg LT(L_{fg})$).

Each link is also assumed to be associated with a cost factor (denoted as $CF(L_{fg})$ for link L_{fg}). The cost factor represents the cost of transferring one packet of data over the link. In this paper, we regard the cost factor as a generic parameter specified according to the characteristics and constraints of the DTN. A commonly used cost factor is the inverse of the link bandwidth ($CF(L_{fg}) = \frac{\mu}{BW(L_{fg})}$), where μ is a constant.

B. CED

CEs are composed from two or more PEs. PEs are events that are generated atomically from the sources. Each event (PE or CE) is associated with a unique identifier. Variable pe_i denotes a PE with ID i, and pe_i^j represents the j^{th} instance of pe_i. Each event instance is also associated with a *Start-Time*

and an *End-Time*. An event instance is said to *occur* within a certain time duration if both the Start-Time and the End-Time of the instance fall within the duration.

As in previous CED schemes [10], [16], [11], our system supports a standard set of event composition operators (shown below). Most of the operators incorporate a time window argument (represented as w) which specifies the maximum duration between any two PE instances that are part of a CE instance. Below, we provide informal descriptions of the operators. Formal descriptions can be found elsewhere [10].

- **and** operator (**and**$(pe_1, pe_2, \ldots, pe_m; w)$): An instance of the CE is detected when at least one instance of every constituent PE occurs within a sliding window of length w. The Start-Time of the CE is set to the *minimum* of the Start-Times of constituent PE instances and End-Time of the CE is the *maximum* of the End-Times of constituent PE instances.
- **seq** operator (**seq**$(pe_1, pe_2, \ldots, pe_m; w)$): A special case of the **and** operator where the PE instances must occur in the *same order* as specified in the parameter list (i.e., $\forall i \in \{1, \ldots, (m-1)\}$, pe_i.End-Time $\leq pe_{i+1}$.Start-Time).
- **or** operator (**or**$(pe_1, pe_2, \ldots, pe_m)$): A CE instance is detected each time an instance of any one the constituent PEs occurs. In contrast to the **and** and the **seq** operators the **or** operator does not require a time window parameter.
- **negation** operator (**!**): Negation operator is used to specifically exclude events in **and** and **seq** operators.

If pe_k appears as an argument in the definition of ce_i, then pe_k is said to be a *constituent event* of ce_i.

C. Problem Statement

Consider a set of m PE sources. Each PE source resides on a DTN node. The DTN may have additional nodes other than those that host PE sources. Every node is assumed to have computation, communication (radio transmission) and storage capabilities.

For each CE, a node of the DTN is designated as its *destination* or *sink* (represented as $V_D(ce_i)$ for the complex event ce_i). This is the node at which the CE is eventually needed. For example, this can be a base station on earth (in case of interplanetary DTNs) or a logistics planning camp (in battlefield DTNs). Each CE is also associated with a *delay tolerance limit* (or simply *delay tolerance*, represented as $\Delta(ce_i)$), which signifies the the maximum detection delay that can be tolerated for that CE. The delay for a CE instance is the difference between the time at which the last constituent PE instance of the CE occurred and the time at which the CE was detected at the destination. The delay of the CE is the maximum of the detection delay over all of its instances.

Given (1) the topology of the DTN (including the EDP, EAP, BW and LT of various links), (2) the location and source of each PE, and (3) the definition of a set of CEs that needs to be detected, their respective destinations, and their individual delay tolerance limits, the problem is to come up with a plan that minimizes the cumulative cost of detecting the set of CEs. The cost of a link L_{fg} under a certain CED plan is the product of its cost factor ($CF(L_{fg})$) and the average number of bytes transferred through the link per unit time. The cumulative cost of a detection plan is the sum of the costs of all links involved in the plan.

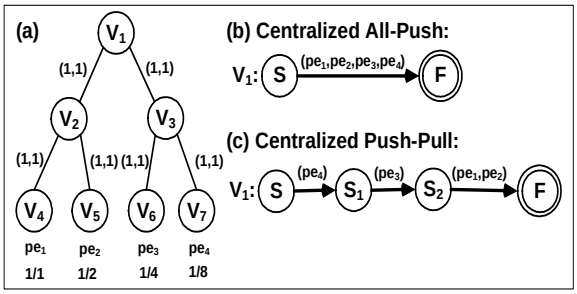

Fig. 1. Centralized CED on DTN

The plan will essentially include three things: (1) where (on which node(s)) the CED process will execute; (2) for each node involved in the CED, which of its constituent events will be pro-actively sent (pushed) to the node, and which will be obtained by the node when needed (pulled); and (3) if a node pulls multiple constituent events, in what order would it be done. A CED plan is usually represented as a set of *finite state machines (FSMs)*. Each node executing the CED process has an associated FSM that specifies the sequence of push and pull operations that are executed by that node.

D. Challenges

With an example, we now explain two existing CED techniques and discuss why they cannot be trivially adopted for DTN settings. Figure 1(a) shows a scheduled contacts DTN with 7 nodes. There are four pe_1 through pe_4, residing on nodes V_4 through V_7 respectively. The PE frequencies are shown in the diagram, where the notation x/y means that x events are generated every y time units. The numbers next to the link indicate their respective cost factors and EDP. Suppose the CEs ce_1 = **and**$(pe_1, pe_2, pe_3, pe_4; 2)$ is to be detected at the node V_1.

The simplest centralized CED approach is to push every instance of each PE as soon as it occurs to the destinations of all CEs that it is part of. The destination checks which set of PE instances result in a CE instance. No other nodes perform any CED tasks. This approach is an adaptation of the CED technique used in active databases and triggers [16], [11]. In this plan, each CE is associated with a single FSM at its respective destination. Figure 1(b) shows the only FSM (at node V_1) of this plan. The other nodes do not perform any detection tasks, and hence do not have associated FSMs.

This approach yields the lowest delay but is very costly—in fact this approach has maximum cost. This is because it pushes instances irrespective of whether an instance has any chance of being a part of a CE instance. In our example, most instances of pe_1 will not become part of any CE because an instance of pe_4 is produced only once in 8 seconds, and the window length is 2. A main drawback of this approach is that

it fails to utilize higher delay tolerance limits for lowering CED costs.

An alternate technique proposed by Akdere et al. [10], attempts to alleviate the problem by selectively pushing certain (usually the cheapest) PEs to the destinations. A destination pulls the other component PEs when it notices (based on PE instances that have arrived) that there is likelihood of a CE instance. The problem is to decide, for each CE, which PE sources will be pulled by its destination and in what order, so that the detection cost is minimized while ensuring that the detection delay does not exceed the specified tolerance limit. The authors provide an optimal algorithm that is exponential-time. They also proposes a heuristic algorithm. Their algorithm employs two kinds of pulls – *single-target pulls* in which the destination sends out a pull request to only one PE source at a time and *multi-target pulls* (or *simultaneous pulls*) in which the node simultaneously pulls from multiple PE sources. Note that in this technique also, the entire CED process essentially occurs at the CE's destination. The plan for a CE will consist of a start state in which certain PEs are pushed to the destination, possibly followed by a sequence of states, each corresponding to a single or a multi-target pull. Figure 1(c) illustrates one such plan for ce_1.

Even this technique suffers from several limitations. First, it can still result in significant degree of wasted communication. In Figure 1(a), one of the lowest-cost centralized CED plan for ce_1 will involve pushing pe_4 to V_1, and V_1 pulling pe_3, pe_2 and pe_1 in that order. Note, however, that for a significant fraction of pe_4 instances, there might not be any pe_3 instances that occur within the time window ($w = 2$). Pushing such events to V_1 will result in higher costs, especially if the cost factor of L_{13} is high. In general, it is better to discard such instances early on. Second, due to frequent and potentially long link disconnections, pulling events by the destination will add significantly to the detection delay. In our example, pulling pe_3 can add up to $2 \times (EDP(L_{1,3}) + EDP(L_{3,6}))$. The factor of two is because the request and response messages can each be preceded by a link disconnection. The problem is exacerbated in situations where PE sources are several hops away from the destination. Considerable delay savings can be obtained by pulling pe_3 from V_3. In general, when the EDP of the links closest to the sources is relatively high, it is beneficial to push the PE to an intermediate node. The destination can then pull from the intermediate node. Centralized CED techniques (including the sophisticated one that allows selective push and pull) preclude such plans.

III. Comet Overview

Comet addresses the above limitations by distributing the CED process among multiple DTN nodes. In other words, multiple nodes may share the task of detecting a CE. A unique aspect of our approach is that it employs *multi-level hierarchical* structures called *CED Trees* (defined later in the section) as the basis for detecting CEs. In this paper, our focus is on the *CED planner*, which devises cost effective CED plans. The plans produced by Comet are fed into an *execution and adaptation engine* that executes the plans and adapts them to cope with various dynamics. The execution and adaptation engine is beyond the scope of the current paper.

As mentioned in Section II, Comet has to provide answers to a set of important and inter-related questions. **(1)** Which sub-CEs of the given CE are to be detected? In other words, how do we (recursively) divide a CE into multiple sub-CEs? **(2)** Where (on which nodes) are the processes for detecting the given CE and each of its sub-CEs going to be hosted? Finally, **(3)** For each CE and sub-CE, which of its component events (PEs or other sub-CEs) are going to be pushed to its hosting node, and which component events are going to be pulled, via single-target and multi-target pulls, and in which order? The goal is to come up with answers to these questions such that the delay tolerance limit of each CE is respected and the cumulative cost of detecting CEs is minimized.

Before discussing the design of our CED planner, we state a few fundamental assumptions that will be used throughout our discussion. First, we assume the planner knows the frequencies of the various PEs of a given CE and the topology of the DTN and the properties of various links (EDP, EAP, BW, LT, CF, and DL). Second, the nodes of the DTN have enough storage to hold all the incoming data until it can be transferred to the next node along its path. Third, once a link becomes active, its EAP and BW are sufficient to transfer all the data in the outgoing buffers of its end nodes. Dealing with resource constraints requires effective prioritization of communication, storage and processing of events, and it is part of our future work.

Fundamental to our CED planner is the concept of a CED tree. The CED tree of a complex event ce_i is composed of ce_i's destination as its root and the source nodes of ce_i's component PEs as it leaves (although a PE source can be a non-leaf node). Comet computes cost-and-delay effective paths (see below) from the source of each component PE to the CE destination. The DTN links and nodes that are part of at least one such path (from a component PE source to ce_i's destination) form edges and the intermediate nodes of the CED tree. A DTN node that lies at the intersection of the paths from two or more PE sources to the CE destination is called a *junction*.

Our system operates on a per-CE basis – i.e., for each individual CE in the system, the CED planner modules are invoked independently to produce a multi-level near-optimal CED plan for that CE. Our CED planner is comprised of three novel components, namely, a *CED tree construction component*, a *multi-level push-pull conversion component*, and a *virtual topology creation component*. The first component employs a novel cost and delay heuristic to create an efficient CED tree for each CE. Our second component addresses the challenges in extending the push-pull conversion-based planning strategy to multi-level CED trees. The third component creates multiple virtual trees for a given CED tree to overcome the potential suboptimality caused by operating at link granularity (see Section IV-C). Comet creates a set of virtual CED trees, executes the push-pull component on each topology, and selects the best plan among them.

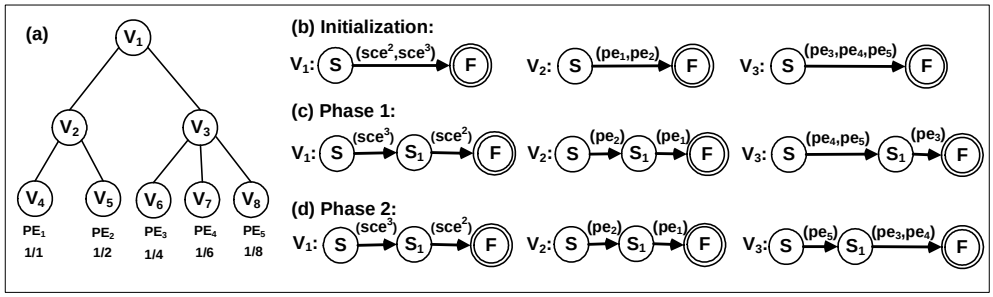

Fig. 2. Illustration of Multi-level Push-Pull Conversion

IV. Decentralized CED Planning in Comet

In this section, we explain each of the three components of Comet's decentralized CED planner.

A. CED Tree Construction and Sub-CE Determination

The first challenge in supporting multi-level CED is to construct an efficient CED tree for each CE. We do this by computing cost-and-delay effective paths from each component PE source to the CE destination. Ideally, the path should minimize both the cost and the delay of transferring PE instances from the source to destination. However, in most practical scenarios it is almost impossible to obtain such paths – DTN links that have minimum cost may not have minimum delay, and vice-versa. We address this problem by assigning weights to DTN links according to a novel *h-function* that combines both cost and the delay characteristics of links. The h-value of a DTN link L_{fg} is computed as $h(L_{fg}) = \alpha \times \frac{CF(L_{fg})}{\text{MAX-CF}} + (1-\alpha) \times \frac{(DL_{fg})}{\text{MAX-DL}}$, where $CF(L_{fg})$ and $DL(L_{fg})$ are the cost factor and delay of L_{fg} respectively, MAX-CF and MAX-DL are the maximum cost factor and maximum delay over all links in the DTN, and α is a weight factor that can be used to adjust the relative importance of cost factor and delay respectively. Notice that the lower the cost factor and delay of a DTN link, the lower the h value.

Once the h-values of all the DTN-links are determined, we use Dijkstra's shortest path algorithm [17] to find the path with minimal cumulative h value from each PE source to the CE destination. The union of these paths form the CED tree. We then determine the set of junction nodes in the CED tree. Each junction in the CED tree may potentially host a sub-CED process. The sub-CE to be hosted at a junction node V_f is determined by applying the same operator as that of the original CE to the set of PEs and sub-CEs that intersect at V_f.

B. Multi-Level Push-Pull Conversion Component

Given a CED tree (original or virtual), this module produces a near-optimal plan (in terms of detection costs) consisting of push-pull schedules at every junction node for detecting the corresponding CE/sub-CE. Our technique starts with a simple plan in which the CED process at every junction node follows a simple 2-state FSM analogous to the all-push plan. This module progressively transforms the FSMs at the junction node through conversion of the corresponding links from push to pull (see Figure 2).

Our scheme operates in two distinct phases. In the first phase, as many links as possible are converted from push to single-target pull without violating the detection delay tolerance limit. In the second phase, we convert as many of the remaining push links as possible to multi-target pulls (i.e., pull them simultaneously with sibling links that already have pull status). The rationale for performing these two phases in this order is that, while converting a push link to a sequential pull always yields higher cost savings, it also substantially increases the CED delay (as much as $2 \times EDP(L_{fg})$ for link L_{fg}). On the other hand, generally, converting a push link to a multi-target pull causes only marginal increase (or in some cases no increase) in detection delay. Our scheme essentially follows a greedy strategy by seeking to maximize cost savings with each conversion in the first phase, and trying to obtain further cost savings, albeit in (relatively) smaller amounts for each conversion, while ensuring that the resulting impact on delay is marginal.

Two important questions need to addressed when converting links from push to single target pulls in the first phase. **(1)** For each junction node, which set of links should be converted from push to pull so that the cost of the plan is minimal and the corresponding delay does not exceed the tolerance? **(2)** If a node has multiple incoming pulls, in which order should they be performed? Since the optimal algorithm to solve question 1 is exponential even for centralized settings (single level CED trees), we adopt a greedy heuristic approach. Since our goal is to minimize cumulative costs, our heuristic is the ratio of cost reduction to the delay increase caused by a push-to-pull conversion. We denote the *cost-to-delay ratio* as CDR, so $CDR(L_{fg}) = \frac{\text{Cost Reduction obtained by converting } L_{fg} \text{ from push to pull}}{\text{Delay increase caused by converting } L_{fg} \text{ from push to pull}}$. Our technique performs push to single target pull conversions in the decreasing order of the links' CDR values until a stage where any additional conversion would cause violation of the specified delay tolerance. If a node has several incoming pull links, the respective component events are pulled in increasing order of their frequencies. The idea is to let the sub-CED process at a node advance only after resolving the most difficult hurdles.

Computing CDR values requires estimation of the cost and delay of a multi-level CED plan. We extend the FSM-based

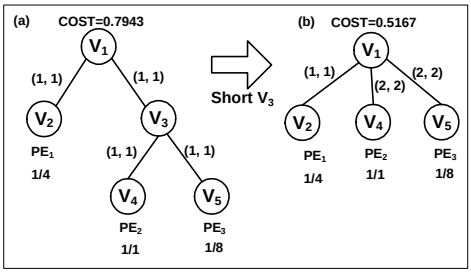

Fig. 3. Virtual Topology Creation via Shorting

cost estimation model [10] for multi-level CED trees. The idea is to use a bottom-up approach to estimate the frequencies of various sub-CEs. This is in turn used to estimate the amount of data transferred per unit time at every link in the CED tree. The cost of a plan is the weighted sum of data transferred per unit time overall links in the tree, the weight being the cost-factor of the link. The delay of a plan is also estimated through a bottom up approach. At each junction node, we estimate the delay of the corresponding CE/sub-CE by analyzing the *critical path* of its FSM (longest sequence of operations), along with the EDP values of the incoming links and the delays of its constituent events. Our technical report provides the mathematical formulation and a detailed discussion of our cost and delay estimation models [18].

In the second phase, our planner checks the links that still have a push status at the end of the first phase to see if any of these links can be converted to multi-target pulls. In order to ensure that delay tolerance limit is honored we enforce the following condition: a link L_{fg} that has push status at the end of phase 1 can be converted to a simultaneous pull with a sibling link L_{fh} only if (1) L_{fh} already has pull status and (2) the push-pull conversion of L_{fg} doesn't violate the delay tolerance limit. We consider the links for conversion in the decreasing order of the estimated cost reduction.

C. Virtual Topology Creation Component

The above multi-level push-pull conversion technique assumes that the junction node of the CED tree hosts a sub-CED process. In most scenarios, executing this component on the original CED tree is sufficient for obtaining a near-optimal plan. However, in certain settings, performing sub-CED at every junction node of the original CED tree will yield plans that are suboptimal irrespective of the combination and order of links that are pushed and pulled.

Figure 3-a gives one such example. In this CED tree, there is one junction node (V_3) other than the destination V_1. On this topology, if the delay tolerance limit is large, our push-pull conversion module will produce the following plan: pe_3 is pushed to V_3; V_3 pulls pe_2; the detected sub-CE (and(pe_3, pe_2)) is pushed to V_1; then V_1 pulls pe_1. The cost in this case is 0.7943 per unit time. In fact, this is the lowest cost plan *if V_3 is forced to detect the sub-event and*(pe_3, pe_2). However, the true lowest-cost plan is to push pe_3 all the way up to V_1, which will then pull pe_1 and subsequently pull pe_2. This yields a cost of 0.5167 per unit time. However, executing our push-pull module on the original CED tree fails to produce this plan.

Our mechanism to circumvent this problem is to create multiple virtual CED trees by selectively eliminating one or more junction nodes through a unique technique called *shorting*. When we short a particular junction node, say V_f, we remove it from the topology and connect each of its children (say V_g and V_h) to V_f's parent node, say V_e. The cost factor of the new link between V_g and V_e is set to the sum of the cost factor of the original link between V_g and V_f and the cost factor of the original link between V_f and V_e ($CF(L_{eg}) = CF(L_{ef}) + CF(L_{fg})$). This is because the cost of transferring a byte of data from V_g to V_e in the original topology is $CF(L_{ef}) + CF(L_{fg})$ if V_f were to just act as a transit node (instead of detecting the sub-CE). Analogously, $EDP(L_{eg})$ is set to $EDP(L_{ef}) + EDP(L_{fg})$ because this is the worst case disconnectivity period between V_g and V_e in the original topology. However, $EAP(L_{eg})$ is approximated as $min(EAP(L_{ef}), EAP(L_{fg}))$ and $BW(L_{eg})$ is approximated as $min(BW(L_{ef}), BW(L_{fg}))$. The reason is that this represents the worst case EAP and bandwidth between V_g and V_e in the original topology. Figure 3-b indicates a virtual topology created by shorting V_3. The numbers next to the links indicate the CF and EDP values, respectively.

Theoretically, we can create virtual topologies by shorting every possible combination of junction nodes and executing the push-pull module on these topologies to yield an optimal plan. However, this is inefficient because it will require us to execute the push-pull module on $\sum_{b=1}^{r} \binom{r}{b}$ where r is the number of junctions in the original CED tree excepting the original destination. Therefore, we adopt a *level-based* strategy. Suppose the original CED tree is of height H. If the tree is *shorted at level q*, all the junctions that are at least q hops away from the destination are eliminated. Note that if a tree is shorted at level 1 we get a single-level tree. If the original CED tree is of height H, Comet generates $H-1$ virtual trees by shorting at levels $H-1$ through 1. The push-pull module is executed on each of these virtual trees in addition to the original CED tree and the lowest cost plan is selected. In our example, if we execute the push-pull strategy on the virtual topology generated by shorting at level 1, which eliminates V_3 (see Figure 3-b), we get the aforementioned lowest-cost plan (pushing pe_3 all the way up to V_1 and then pulling pe_2 followed by pe_1).

V. Experimental Evaluation

A. Experiment Setup

We have implemented both *Comet* and our DTN simulator in Java. The DTN simulator simulates the DTN model described in Section II-A. The simulator contains a number of DTN nodes, each of which connects to its neighbors according to a given schedule. Each DTN node can be either a PE source, a CE sink, or a junction node, depending on how the CED tree is constructed. If the node is a PE source, it generates PE instances according to a Poisson distribution. We use the

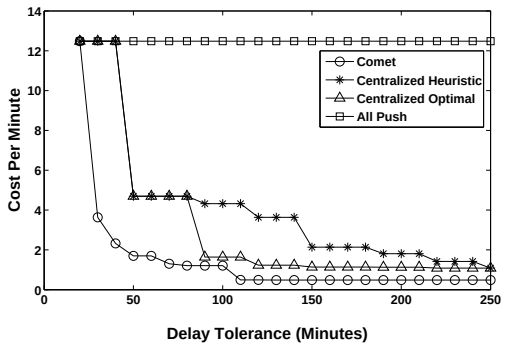

Fig. 4. Performance of the four algorithms with nonuniform cost and uniform latency per link.

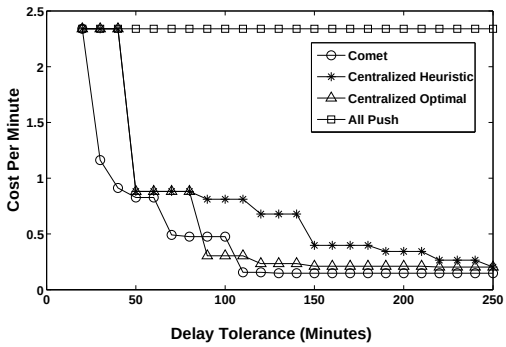

Fig. 5. Performance of the four algorithms with uniform cost and uniform latency per link.

Zipfian distribution to generate the PE occurrence frequencies. Each DTN node is also capable of executing the sub-CED plan, which is represented as a finite state machine.

In all of our experiments, we assume that the DTN links are reliable when they are in operation. Also, recall that we assume the expected active period (EAP) of all links is sufficiently long to transmit all data in the buffer of the sending node. We will focus on two major properties of the DTN link – Bandwidth and Expected Disconnection Period (again, see Section II-A). Our planner and DTN simulator support different models for bandwidth and EDP. However, for simplicity, we use three categories of bandwidths: low bandwidth (128 Kbps), medium bandwidth (256 Kbps) and high bandwidth (1.2 Mbps). We define the cost factor as $\frac{packet\ size}{bandwidth}$. For EDP, we also use three categories: low EDP (30 seconds expected disconnection period), medium EDP (2.5 minutes expected disconnection period), and high EDP (5 minutes expected disconnection period).

B. Results

For each experiment, we show the results of four different algorithms. The baseline plan is *All-Push*, where all events are pushed to the sink immediately. *All-Push* always satisfies any delay restriction for which there exists a feasible plan. The *Centralized Optimal* plan is the one suggested as optimal in the work by Akdere et al. [10]. Note that it is optimal only for solutions in which pulling of events occurs only at the sink, so Comet, with its multi-stage nature, can outperform *Centralized Optimal*. The *Centralized Heuristic* plan is our adaptation of the heuristic algorithm suggested in [10]. Finally, Comet is our novel multi-stage heuristic plan described in Section IV.

Figure 4 shows results of cost for the aforementioned four algorithms. The delay tolerance ranges from 0 to 250 minutes. The topology in this experiment is such that the EDP is high (5 minutes) for all links, and the bandwidth per link is 128 Kbps on all links connected to the sink, 1.2 Mbps on all links connected to the sources, and 256 Kbps on all other links. Note that for delay restrictions smaller than 16.5 minutes, there is no feasible solution, even with *All-Push*.

The results clearly show that Comet is superior to the other three algorithms. Comet has a cost that is 89% less than *All-Push*, 66% less than *Centralized Heuristic*, and 56% less than *Centralized Optimal*. Specifically, as expected, *All-Push* fails to filter PEs and so incurs a large cost across various DTN links due to transmission of PEs that cannot be part of any CE. While the other two algorithms—*Centralized Optimal* and *Centralized Heuristic*—are able to filter out some PEs. Comet is superior to both because its multi-stage nature allows PEs to be pulled from nodes closer to the source; i.e., from the leaves to the interior nodes, which filters out additional PEs. Again, *Centralized Optimal* and *Centralized Heuristic* only pull events at the destination. Note that creating a multi-stage optimal algorithm is infeasible because it is exponential in the number of links.

A comparison of the *Centralized Heuristic* to *Centralized Optimal* shows that while *Centralized Heuristic* produces identical results in some cases, it is inferior in other cases. This is because *Centralized Heuristic* first examines sequential pulls, which can result in searching of a subspace of the solution space in which the optimal plan does not occur. *Centralized Optimal* enumerates all possible plans in which pulls are performed at the sink.

Figure 5 shows results for a similar experiment as was shown in Figure 4, except that the cost per link is uniform. The results are similar in many cases, but there is a range of delay restrictions–85 to 100—in which Comet has a higher cost than *Centralized Optimal*. This occurs because when the cost per link is uniform, the benefit of pulling PEs close to the source is lower. As *Centralized Optimal* explores more of the (plan) solution space, it can and does perform better for this small range of delay restrictions. Future work will address this issue; briefly, we plan to explore the potential of concurrent pulls when the delay tolerance is modest. The timing of pull requests may be reassigned so that the event sources with similar frequencies will be pulled concurrently depending on the given delay tolerance. We will also consider re-scanning the overlay tree topology to balance the sub-CE latencies for different subtrees, which will eventually fully utilize the potential concurrency in detecting CEs.

Figure 6 shows results for the same two experiments as discussed above, except that the link latency is now nonuniform.

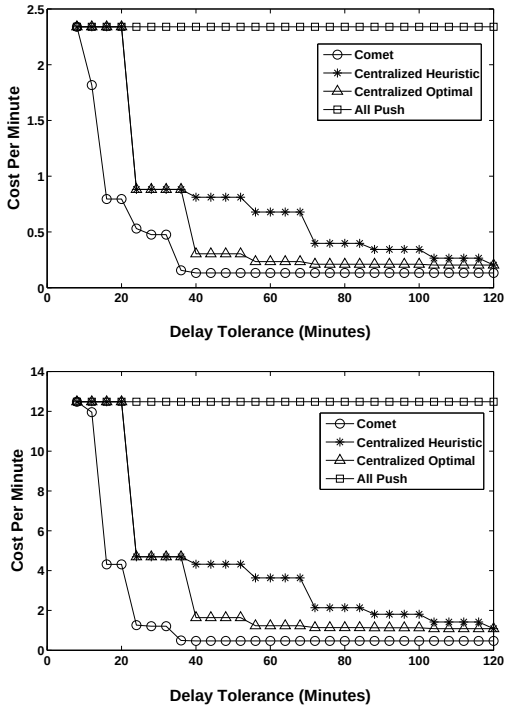

Fig. 6. Performance of the four algorithms with nonuniform latency per link, for both uniform (top) and nonuniform (bottom) cost per link.

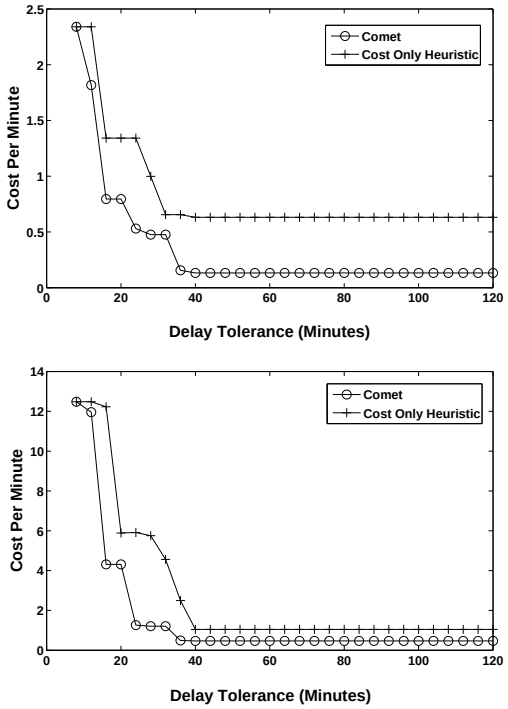

Fig. 7. Performance of Comet with different heuristics. The top figure has uniform cost per link, and the bottom figure has nonuniform cost per link.

The links connected to the sink have an EDP of 5 minutes, an EDP of 30 seconds for the links connected to the sources, and an EDP of 2.5 minutes for all other links. Essentially, this experiment shows similar, if not quite as pronounced, results to Figure 4.

Next, Figure 7 shows results for the same two experiments as discussed in Figure 6, except that the ordering of FSM status changes are determined using only cost rather than the ratio of cost to latency. Note that in this experiment we compare only the two versions of Comet.

This experiment makes it clear that it is better to use the ratio of cost to latency for ordering potential status changes in Comet. On one hand, using purely cost, irrespective of the change in latency, may cause Comet to choose pull operations that can cause significant plan latency increases and also leads to fewer pull operations elsewhere in the plan due to the delay tolerance. On the other hand, using the ratio of cost reduction to latency better balances the change of both cost and latency. It also can leverage the potential of pull concurrency, which can in turn lead to cost reduction with only a small latency penalty.

Figure 8 shows results of the four algorithms on a skewed topology, in which the degree of junction nodes varies from 1 to 3. The EDP is set to high (5 minutes) for all links, and the bandwidth per link is 128 Kbps for all links connected to the sink, 1.2 Mbps for all links connected to the sources, and 256 Kbps on all other links. Again Comet is superior to all other algorithms, even with such a skewed topology. On average, Comet is 61% less than *Centralized Optimal*, 69% less than *Centralized Heuristic*, and 89% less than *All-Push* in term of cost per minute.

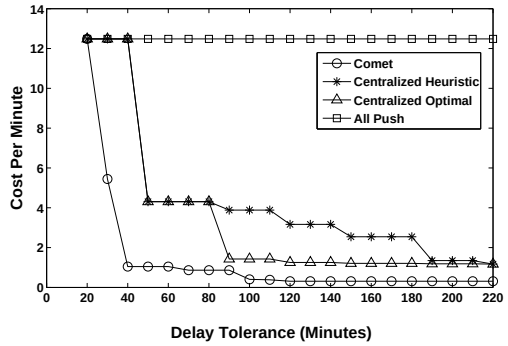

Fig. 8. Performance of Comet where the degree of the junction nodes is varied (randomly) from 1 to 3.

Figure 9 shows the impact of the depth of CED tree on the cost of the detection plan. When the depth of CED tree increases, the cost decreases. This is due to the multi-stage sub-CE detection of Comet; recall that it allows the PEs to be pulled from junction nodes closer to the source. This not only alleviates the load at the links connected to the sink, where the bandwidth is usually limited, but also significantly removes the unnecessary latency due to the long turnaround time of the pull request and reply between the sink and source. At times when there is no PE satisfying the pull request, the penalty is limited because of the relative short latency between the junction node and the source. Note that at the

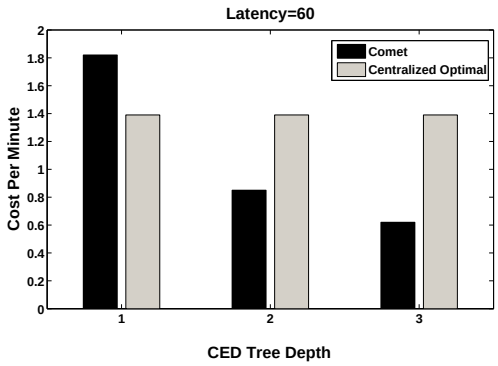

Fig. 9. Performance of Comet with different CED tree depth.

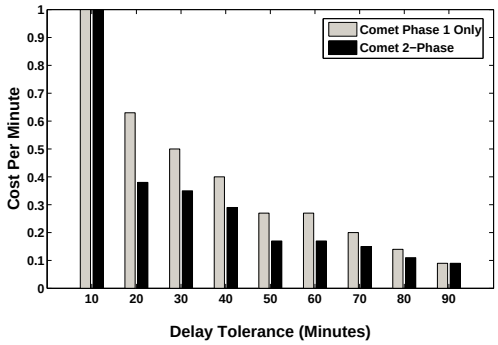

Fig. 10. Benefit of using two-phase algorithm in Comet.

same time, the cost of centralized optimal remains constant, because centralized detection plans do not utilize the junction node to further reduce the plan cost and latency.

Figure 10 shows the benefit of the 2-phase algorithm of Comet. The concurrent pull phase of Comet (which is the second phase) further explores the concurrency of pull operations, especially when the delay tolerance is modest. Most of the time, the 2-phase algorithm results in a significant cost reduction compared with the algorithm with only conversion of pushes to single target pulls (which is the first phase). The second phase further reduces the cost by converting more pushes to pulls, but without a significant latency penalty. Note that in this figure, when the delay tolerance is 10 and 90 minutes, there is no difference between the single phase and two-phase algorithms. This is because (1) at the tolerance of 10 minutes, the only available plan is to push all events to the sink; and (2) at 90 minutes, there is nothing for the concurrent pull phase to improve, because the first phase has already converted all the available push operations.

VI. Discussion

As our experimental study has demonstrated, Comet produces high quality plans in most cases. However, there is room for further improvement. We have identified two specific avenues for improvement. First, in the push-pull conversion module, multi-target pulls are explored only when no more push links can be converted to single target pulls. This can be problematic in certain situations – a decision to convert a particular link from push to single-target pull may preclude several conversions from push to multi-target pulls in the second phase, which collectively might have yielded higher cost savings. An attractive option is to identify and reverse problematic single-target conversions in the second phase. Second, currently, if we short the CED tree at a particular level all the junctions at that level are eliminated. A better strategy is to operate at a finer granularity and short only the set of junction nodes that are contributing to higher detection costs. Since an exhaustive search of all possible combinations of junction nodes is highly inefficient (see Section IV-C, we need to design a simple heuristic for checking whether shorting a particular node will lead to higher cost savings.

In addition to these immediate improvements, there are a number of open challenges that require significant enhancement of our current system. First, we assume that the various nodes and links in the system have enough storage and bandwidth resources to hold and transfer all event instances. These assumptions may not be valid for highly resource constrained environments. In such scenarios, the planner will have to incorporate smart and effective prioritization and load-shedding mechanisms. Second, we need to design incremental strategies to deal with environmental dynamics such as additions/deletions of CEs and variations in link and node characteristics. Third, many domains DTN nodes are powered by batteries. For these domains, it is important to consider power consumption as an additional factor during CED planning. Addressing these challenges is a part of our future work.

VII. Related Work

CED originated in the field of active database systems as a mechanism to respond automatically to events that are taking place either inside or outside of the database system [11]. Current work on CED has focused on two main issues, namely, reducing the computational overheads at the server [12], [13] and reducing the communication costs [10]. The plan-based CED technique [10] reduces the communication overheads of CED by intelligently pushing and pulling PEs. There are several crucial differences between Comet and these existing systems. Most importantly, the above techniques are centralized in the sense that the entire CED process occurs at a single node. Comet on the other hand is based on multi-level CED paradigm and it enables sharing of CED tasks among multiple nodes. Chandramouli et al. [19] study the problem of accurately estimating latency in distributed event processing systems. Distributed data stream processing [20], [21], [22], [23], [24], [25], [26] is another area that is closely related to our work. However, as Akdere et al. [10] have noted, data stream processing systems have to rely exclusively on push-based data transfer. On the other hand, CED systems have the flexibility of utilizing both push and pull data transfer modes thus providing an additional dimension for optimization. It is also noteworthy that most of the current distributed data stream processing systems have been assuming a traditional,

continuously-connected network. Our work also bears similarity to poller-pollee model used for data/status aggregation in wireless sensor networks [27]. Research on data aggregation has mostly focused on reducing the number of pollers and its impact on false alarm rates. Our work is somewhat orthogonal in that it addresses planning with respect to ordering and scheduling of push and pull operations (proactive and reactive modes). It will be interesting to study the interactions between these two problems.

DTN has been an active area of research for the past few years [6], [5]. The major focus is designing effective routing and message dissemination schemes [6], [7], [8]. Typically, routing strategies exploit connectivity patterns, node mobility patterns, packet replication, and social affinity for achieving effective packet delivery [5], [28], [3]. Unfortunately, research on building applications and systems on DTNs has thus far been confined to simple web applications, distributed file systems, and caching [29], [30], [9]. This paper is a step towards closing this critical gap.

VIII. Conclusion

Current centralized CED techniques have significant limitations that make them ineffective for multi-hop DTN environments. In this paper, we present Comet, which, to the best our knowledge is the first decentralized multi-level CED planner in which the multiple DTN nodes share CED tasks. Comet's planner is characterized by three novel techniques. First, it constructs cost-and-delay efficient CED trees using a unique h-function. Second, it incorporates a two-phase push-pull conversion heuristic that employs both single-target and multi-target pulls to progressively lower CED costs. Third, a unique technique called shorting is designed for creating virtual topologies. Shorting creates virtual CED trees by eliminating junction nodes at various levels of the CED tree, thereby countering scenarios in which detecting sub-CEs at every junction leads to suboptimality. Through extensive experimental evaluation, we have shown that in most cases Comet produces significantly better plans than existing centralized CED mechanisms.

References

[1] C. Rigano, K. Scott, J. Bush, R. Edell, S. Parikh, R. Wade, and B. Adamson, "Mitigating naval network instabilities with disruption tolerant networking," in *Proceedings of MILCOM*, 2008.

[2] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, "Lowcost communication for rural internet kiosks using mechanical backhaul," in *Proceedings of MOBICOM*, 2006.

[3] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang, "Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing," in *Proceedings of MOBICOM*, 2007.

[4] V. C. et al., "Delay-tolerant network architecture," *IETF RFC 4838, informational, April 2007*.

[5] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "Maxprop: Routing for vehicle-based disruption-tolerant networks," in *In Proc. IEEE INFOCOM*, 2006.

[6] S. Jain, K. R. Fall, and R. K. Patra, "Routing in a delay tolerant network," in *SIGCOMM*, 2004.

[7] Q. Li, S. Zhu, , and G. Cao, "Routing in socially selfish delay tolerant networks," in *IEEE INFOCOM*, 2010.

[8] W. Gao and G. Cao, "On exploiting transient contact patterns for data forwarding in delay tolerant networks," in *ICNP*, 2010.

[9] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, "Supporting Cooperative Caching in Disruption Tolerant Networks," in *ICDCS*, 2011.

[10] M. Akdere, U. Cetintemel, and N. Tatbul, "Plan-based complex event detection across distributed sources," in *Proceedings of VLDB*, 2008.

[11] N. W. Paton and O. Díaz, "Active database systems," *ACM Computing Surveys*, vol. 31, 1999.

[12] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. S. Candan, "Runtime semantic query optimization for event stream processing," in *ICDE*, 2008.

[13] E. Wu, "High-performance complex event processing over streams," in *In SIGMOD*, 2006, pp. 407–418.

[14] L. Brenna, J. Gehrke, M. Hong, and D. Johansen, "Distributed event stream processing with non-deterministic finite automata," in *DEBS*, 2009.

[15] P. R. Pietzuch, B. Shand, and J. Bacon, "A framework for event composition in distributed systems," in *MIDDLEWARE*, 2003.

[16] E. N. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha, S. Parthasarathy, J. B. Park, and A. Vernon, "Scalable trigger processing," in *Proceedings of ICDE*, 1999.

[17] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, 1959.

[18] J. Chen, L. Ramaswamy, D. K. Lowenthal, and S. Kalyanaraman, "Comet: Decentralized complex event detection in delay tolerant networks," Department of Computer Science, University of Georgia, Tech. Rep. UGA-CS-TR-11-001, July 2011. [Online]. Available: http://www.cs.arizona.edu/people/dkl/comet-tr.pdf

[19] B. Chandramouli, J. Goldstein, R. S. Barga, M. Riedewald, and I. Santos, "Accurate latency estimation in a distributed event processing system," in *ICDE*, 2011.

[20] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the borealis stream processing engine," in *Proceedings of CIDR*, 2005.

[21] B. Gedik, H. Andrade, and K.-L. Wu, "A code generation approach to optimizing high-performance distributed data stream processing," in *CIKM*, 2009.

[22] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah, "Telegraphcq: Continuous dataflow processing for an uncertain world," in *Proceedings of CIDR*, 2003.

[23] M. F. Mokbel and W. G. Aref, "Sole: scalable on-line execution of continuous queries on spatio-temporal data streams," *VLDB J.*, vol. 17, no. 5, 2008.

[24] P. R. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. I. Seltzer, "Network-aware operator placement for stream-processing systems," in *Proceedings of ICDE*, 2006.

[25] S. Babu and J. Widom, "Streamon: An adaptive engine for stream query processing," in *SIGMOD Conference*, 2004.

[26] Y. Ahmad, O. Papaemmanouil, U. Çetintemel, and J. Rogers, "Simultaneous equation systems for query processing on continuous-time data streams," in *ICDE*, 2008.

[27] C. Liu and G. Cao, "Distributed monitoring and aggregation in wireless sensor networks," in *INFOCOM*, 2010.

[28] W. Gao, Q. Li, B. Zhao, and G. Cao, "Multicasting in delay tolerant networks: A social network perspective," in *ACM Mobihoc*, 2009.

[29] A. Balasubramanian, B. N. Levine, and A. Venkataramani, "Enhancing interactive web applications in hybrid networks," in *MOBICOM*, 2008.

[30] M. J. Demmer, B. Du, and E. A. Brewer, "Tierstore: A distributed filesystem for challenged networks in developing regions," in *FAST*, 2008.