Towards Advanced Event Monitoring Services on Decentralized and Delay Tolerant Networks

by

Jianxia Chen

(Under the direction of Lakshmish Ramaswamy and David Lowenthal)

Abstract

Event monitoring services are rapidly gaining importance in many application domains ranging from real time monitoring systems in production, logistics and networking to complex event monitoring in finance and security. However, the current event monitoring services do not have the capabilities needed for emerging domains of applications. This dissertation is devoted to study and address the challenges involved in providing event monitoring services on decentralized and delay tolerant networks.

First, we consider the problem of event aggregation and redundancy elimination in decentralized broker overlays. We propose two systems for efficient event aggregation and redundancy elimination. The first system, Agele, presents our ideas for event gatherer, a designated broker in the routing graph that acts as a proxy sink for all messages of a particular event. The second system, Caeva, is built on Agele. Caeva exhibits three novel features: multiple distributed aggregators, adaptive aggregator placement and customized subscriber notification schedule.

Second, we consider the complex event detection on delay tolerant networks. The existing work on complex event detection employs a centralized approach, whose limitations are exacerbated when the underlying environment is delay tolerant networks with long latency and intermittent connection. Hence, the event detection system has to be extensively redesigned for the adaptation

to underlying environment. We propose a novel multi-level framework called Comet for complex event detection services on delay tolerant networks.

The evaluation has demonstrated that our solutions to the challenges in the advanced event monitoring services are effective and efficient.

INDEX WORDS: Event Monitoring, Event Aggregation, Redundancy Elimination, Complex Event Detection, Delay Tolerant Network, Heuristic Algorithm

TOWARDS ADVANCED EVENT MONITORING SERVICES ON DECENTRALIZED AND DELAY TOLERANT NETWORKS

by

JIANXIA CHEN

B.E., Xiamen University, 2002

M.E., Xiamen University, 2005

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2011

TOWARDS ADVANCED EVENT MONITORING SERVICES ON DECENTRALIZED AND DELAY TOLERANT NETWORKS

by

JIANXIA CHEN

Approved:

Major Professors:	Lakshmish Ramaswamy David Lowenthal
Committee:	Suchendra Bhandarkar Kang Li Shivkumar Kalyanaraman

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2011

DEDICATION

To my dear parents and my beloved girlfriend Jiran for their constant love and support.

ACKNOWLEDGMENTS

First, I would like to sincerely thank my advisor Dr. Lakshmish Ramaswamy for his vision, guidance, understanding and patience during my Ph.D program. I would also greatly thank my co-advisor Dr. David Lowenthal for his suggestion, encouragement, support and humor. I would also like to express my gratitude to my committee members: Dr. Suchendra Bhandarkar, Dr. Kang Li and Dr. Shivkumar Kalyanaraman for their precious time and invaluable advices.

This dissertation work was funded by the Dissertation Completion Award from the Graduate School, The University of Georgia.

Table of Contents

LIST OF FIGURES

List of Tables

CHAPTER 1

INTRODUCTION

Event monitoring services are the services that monitor the real-world events which are of interest to the users, and deliver the event notifications to the users. Real-world events are represented as data (or messages) in the event monitoring services. With increased commodity use of mobile devices, and the popularity of collaborative social oriented applications, the event monitoring services are rapidly gaining importance in many emerging domains, such as production, logistics, finance and security etc. Currently, the event monitoring services take the form of publish-subscribe (pub-sub) [1, 2, 3, 4, 5, 6, 7, 8, 9], distributed stream processing [10, 11, 12, 13, 14, 15, 16, 17] and complex event processing [18, 19, 20, 19, 21, 22]. The current event monitoring services are usually based on the decentralized broker overlay, which is built on top of the traditional TCP/IP based network. Examples of such overlay networks are Gnutella [23], CAN [24], Chord [25] and Pastry [26].

The pub-sub event monitoring services consist of publishers, subscribers and the pub-sub system. The publishers publish events to the systems, the subscribers subscribe the events of interest. The pub-sub system offers the event monitoring services to the subcribers, such that the published events that match the interest of the subcribers will be delivered to the subscribers. The pub-sub system is loosely coupled and uses an asynchronous communication model, which makes it particularly well-suited for large-scale distributed applications. The pub-sub systems generally are categorized as topic-based [6], type-based [27], type- and attribute-based [7] and contented-based [3].

Event monitoring services also take the form of distributed stream processing, in which the events are pushed and processed by different operators in the in the overlay network. Unlike traditional database management system, where the “active” users pull the “passive” data, in the distributed stream processing systems, the “active” event data are processed by relatively “passive” operators. Research in the distributed stream processing includes operators and description languages [28], centralized streaming processing engines [10], load balancing [29], fault-tolerance [30], high availability [31] and revision processing [32] etc.

Recently, the complex event detection (CED) services have become a key capability of emerging domain of event monitoring applications. Complex events (CEs) are composed from multiple atomic, possibly geographically distributed primitive events (PEs) [20]. CED services is capable of monitoring correlated events. Current work on CED has focused on two main issues, namely, reducing the computational overheads at the central processing site [18, 19] and reducing the event communication costs on the broker overlay [20].

Existing event monitoring services have the following limitations:

First, redundancy in the event messages is not taken into consideration, which results in significant overhead to both the event monitoring system and the end-user devices. Most existing event monitoring systems make an implicit assumption that the published event data is clean, complete, and accurate (i.e., the event data is devoid of noise). While the assumption may be valid in applications where the publications are electronically generated (i.e, the publishers are gadgets such as servers and different kinds of sensors), this is hardly the case in systems with human participants. In community/social group-oriented event monitoring services, noisy events are almost a given. This noise can take various forms, such as redundant events, incomplete event messages, inaccurate event messages, and even events generated with malicious intent. Redundant event messages and messages containing incomplete (partial) information about the corresponding events are among the most common forms of noise in applications with human participants. As more and more human participants in the social networks become “human sensors”, who monitor and report events of interest. Upon the observation of certain interested event, large group of users

tend to send out event messages which contain significant amount of redundant information. The resulting high volume of redundant event messages will overwhelm the event monitoring systems and the end-user devices as well, especially those devices with limited resources, such as power and band-width. Ideally, superfluous messages should be filtered by eliminating redundant event messages and aggregating event messages containing partial information. However, doing so poses significant challenges both from data management and distributed systems perspectives.

Second, the overhead of existing complex event detection services is prohibitively expensive, when the underlying network is delay tolerant network. With increased commodity use of mobile devices, delay-tolerant networks, or DTNs, are necessarily becoming more commonplace. DTNs exist outside of the Internet; DTN links are characterized by long delays, frequent interruptions, and high error rates [33]. Examples of such networks include battlefield [34, 35], rural [36, 37, 38], vehicular [39, 40], and interplanetary networks [41].

Research on DTNs has primarily focused on routing data from source to destination [40, 42]. Unfortunately, there is little research on building applications that are important and appropriate for these networks. Complex event detection is one such application whose importance and applicability transcends the diversity of different DTN domains. As noted by several researchers, complex or composite event detection (CED) is one of the fundamental components of an event monitoring application. For example, traffic incident and congestion monitoring are attractive for vehicular networks, while monitoring seismic and meteorological conditions are important for interplanetary networks.

Existing work in CED services rely, explicitly or implicitly, on centralized processing of PEs. Unfortunately, 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.

1.1 Dissertation Contributions

To address the limitations and problems of current event monitoring services, this disseration makes the following contributions:

In the first contribution, we present *Agele*, an event monitoring service that embodies our ideas towards reducing superfluous messages from potentially distinct event publishers in semi-real time. *Agele*'s architecture is based upon a decentralized broker overlay, where the message brokers interact with one another in peer-to-peer (p2p) fashion. The twin design goals of *Agele* are to minimize the message load in the overlay and to simultaneously minimize the latency overheads of subscribers. These goals are achieved by elimination and aggregation of event messages as they travel through the overlay from their publishers to the subscribers. We introduce the concept of *event gatherers*, which are broker nodes that identify and eliminate redundant event messages, as well as temporarily hold and merge partial event messages. We show that in order to achieve our goal of minimizing message load in the system, the event gatherer for a set of related subscription predicates should be located at the *graph center* of the corresponding routing acyclic graph. We present a novel decentralized algorithm for determining the graph center of a acyclic graph. An important feature of our algorithm is that it does not need a global view of the overlay, and it only relies upon message exchanges between neighboring nodes in the overlay. Further, our algorithm is efficient and provably accurate. We perform experiments to study the viability of our ideas. *Agele* reduces the message load by over 90% in some cases and over 60% in most. Furthermore, the overhead to subscribers can be kept typically to around 25%, and results in a real-world system with stringent bandwidth constraints would likely be much smaller.

In the second contribution, in order to further achieve scalability and load balancing for event aggregation and redundancy elimination in event monitoring services, in the extended work of *Agele*, we describe the design, implementation, and performance of *Caeva*, which is a decentralized, dynamic, and configurable event monitoring system that handles redundant and partial events. *Caeva* uses a collaborative broker overlay to eliminate redundant messages and merge same-event messages. By performing this task at the brokers, *Caeva* avoids placing this burden on the sub-

scribers, who may be resource constrained in terms of power or bandwidth. To operate effectively at a large scale, *Caeva* must address two key problems. First, aggregation must be decentralized, dynamic, and adaptive to achieve good performance, and the key to achieving this is developing an effective algorithm for placing aggregators within the broker overlay. Second, the ability of subscribers to control the inherent tradeoff between degree of aggregation and latency of notification is critical for usability. We present a collaborative event aggregation and redundancy elimination technique, in which event messages are aggregated in multiple levels and at multiple aggregators. Our technique includes decentralized protocols to coordinate the actions of various aggregators of an event so that subscribers receive notifications with low delay. We design and implement a distributed aggregator placement algorithm that continuously adapts to event message publication patterns with the aim of minimizing the message load within the overlay. In addition, we also develop an efficient notification scheme for supporting subscriber-specified notification cycles. This is done through a unique combination of upstream propagation of individual notification schedules and selective downstream propagation of aggregated messages. We study the benefits and overheads of our scalable, decentralized mechanisms through series of experiments with particular attention to the broker overlay and the resource-constrained subscribers. The results demonstrate that the message load in *Caeva* system can be over 70% less than *Siena* [3], a similar system that does no message elimination; furthermore, the number of messages arriving at subscribers is up to a factor of 2 lower in *Caeva* than in *Siena*.

In the third contribution, this dissertation contributes a novel, multi-level framework, called *Comet*, for efficient and scalable complex event detection (CED) in delay tolerant networks (DTNs). To the best of our knowledge, Comet is the first system that 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 exponential, we propose a novel cost-latency sensitive heuristic algorithm. Given a CE definition and DTN topology, we start with a lowest latency and highest cost plan in which all PEs and sub-CEs are pushed to the next level. Then, we progressively change the status of individual links at various

levels from push to pull until the delay tolerance does not permit any more changes. Our algorithm adopts a greedy strategy and chooses links that provide high utility value in terms of the ratio of cost savings to CED delay. In order to exploit multi-target pulls to further lower communication costs, when single-target pulls are no longer possible due to the delay tolerance, our algorithm switches to simultaneous pull mode. In this mode, we consider changing remaining links with push status to simultaneous pulls with existing pull status links. This reduces the costs with typically only a marginal increase in delay. 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*. Our link-based heuristic algorithm is executed on these virtual topologies, and the plan that yields the minimum cost is selected. The overall event monitoring cost can be further reduced by exploiting the shared events among different types of CEs. Multiple types of CEs may share common PEs. Using a novel heuristic algorithm, Comet reconstructs the CED trees with shared common PEs, so that the shared PEs can be detected as sub-CEs at a common junction node. Hence the cost of detecting the sub-CEs can be shared by multiple types of CEs, which significantly reduces the overall detection cost for all types of CEs. 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 [20]. Comet can further reduce the overall cost by more than 30% when the shared PEs are leveraged. Moreover, in most topologies, Comet outperforms all other techniques, often by similar margins. This includes both skewed topologies and topologies with increasing depth.

1.2 Organization of the Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 provides the background and challenges of this dissertation. Chapter 3 addresses the problem of event aggregation in the

advanced event monitoring services with decentralized broker overlay. Chapter 4 further improves the solution with a more scalable and customizable distributed approach. Chapter 5 is dedicated to the complex event detection in advanced event monitoring services, with a special focus on the delay tolerant networks, followed by Chapter 6 with literature review on advanced event monitoring services. Finally, Chapter 7 concludes the dissertation.

CHAPTER 2

BACKGROUND AND CHALLENGES

In this chapter we briefly discuss the fundamentals of advanced event monitoring services, which includes event aggregation, redundancy elimination, complex event detection (CED) and delay tolerant networks (DTNs). We also formally state the problem of CED on DTNs and explain why the existing CED techniques are not appropriate for DTNs.

2.1 EVENT AGGREGATION AND REDUNDANCY ELIMINATION

Recently, community/social group-oriented event services such as twitter [43] are gaining popularity. In these applications, the members of a (possibly ad-hoc) community or social group collaboratively report and receive events that may be of interest to that group. Observe that this is a pub-sub system in which participants (both publishers and subscribers) are human. Generally, the application provides a subscription mechanism through which a participant can specify the events of interest. Participants who notice a particular event that may be of interest to the community report it to the system using their communication device (desktops, laptops, mobile gadgets, etc.).

In such applications, several participants may notice an event simultaneously (or within a short duration of time). However, some or all of these participants may only have partial information about the event. The event messages published by these participants would naturally contain partial information. Also, individual messages may contain varying degrees of information about the event. Since several participants may individually publish an event to the system, it can lead to redundant event messages. Event message redundancy can be of two forms. First, two event messages may contain the exact same information. This *direct* form of redundancy is generally referred to as *exact duplicates*. However, there is another, subtler form of redundancy. The information

contained in an event message might have already been provided in “bits and pieces” by a set of publishers. In other words, a set of previous messages *cumulatively contains* all the information that the current message is carrying. We call this *indirect* redundancy. As an example, consider the example of a collaborative traffic incident report system, wherein participants report traffic incidents that they notice, which would then be delivered to set of subscribers. In such a system, many participants might report a traffic incident. While all messages pertaining to an accident would contain its location, some of them might provide details about the accident such as how many cars are involved, whether is a fire and whether there are casualties. Some may contain information that is already known from earlier messages.

The simplest strategy for dealing with redundant and partial event messages would be to relegate these responsibilities to the subscribers. This *hands-off* approach, however, is fraught with several drawbacks. First, the subscriber devices may not have the computational and communication capabilities to deal with redundant and partial event messages. For each actual event, subscribers may receive several messages that may overwhelm low-end devices, thus causing them to drop legitimate and important messages. Second, even if the devices were capable of handling the message load, this approach would lead to significant bandwidth overhead, and, in case of mobile devices, corresponding battery-power drain. Third, it would also lead to high communication overhead within the pub-sub system, thereby affecting its scalability and efficiency. Fourth, this approach assumes that the subscriber devices have the software necessary for for performing aggregation and elimination, which raises practicality concerns due to the vast heterogeneity among the subscriber devices.

Thus, it is desirable to have an underlying system that aggregates event information and eliminates redundant messages so that a single (or at most a few) consolidated event messages are delivered to the subscribers. An alternative would be to perform these operations at centralized brokers (one broker per event) of the overlay [44]. However, the centralized brokers can quickly become overloaded. Further, relaying each published message to a centralized broker causes high messaging overheads within the overlay. Thus, in order to achieve scalability, the task of aggre-

gating messages of an event should be shared by multiple brokers, and the set of brokers involved in aggregation should adapt to message publication patterns. In addition, the subscribers should be able to choose the degree of consolidation as per their needs and resource availabilities.

2.1.1 Research Challenges

Designing a distributed broker network that delivers consolidated non-redundant event messages to the subscribers poses a number of distributed systems problems. First, since the messages corresponding to an event may be published by several publishers, event messages may enter the broker overlay through many different brokers, and, depending upon the routing scheme employed by the system, the event messages might follow non-overlapping (or barely overlapping) paths to the subscribers. Hence, individual brokers in the overlay might not even be aware of the existence of multiple (redundant and incomplete) event messages. Second, the individual messages pertaining to an events might be published over a certain period of time, which necessitates storage and maintenance of aggregated information about different events. The questions in this regard are where (on which broker node(s)) would this information be maintained and for how long? Third, consolidating messages pertaining to an event implies that there would be an inherent delay in notifying the subscribers of the information that has been published about it. Managing the tradeoffs between the extent of aggregation and the delay in communicating published information about events to the respective subscribers is another challenge. Fourth, failure of individual brokers may result in loss of aggregated event information in addition to service disruptions. Protecting loss of aggregated event information in the face of broker node failures is yet another problem.

2.2 Delay Tolerant Networks

Delay Tolerant Networks (DTNs) are 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, asymmetric data rates, and high error rates. DTNs operate on a store-and-forward paradigm where a node stores data it receives

until a link that can carry the data forward 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*. As the name suggests, in scheduled-contacts DTNs, 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.

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 or contact is the intermittent connection between two nodes. The intermittent link between nodes V_f and V_g is represented as L_{fg}. Each link is associated with four 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, $EDP(L_{fg}) \gg LT(L_{fg})$ and $EAP(L_{fg}) \gg LT(L_{fg})$. Several routing techniques have been proposed for scheduled-contacts DTNs [40, 42].

2.3 Complex Event Detection

As mentioned in the introduction, complex events (CEs) are composed from two or more primitive events (PEs). PEs are events that are generated atomically from the sources. For example, the surface temperature at a particular location exceeding $100°\mathrm{F}$ is a PE that is produced by a temperature sensor at that location. Each time the surface temperature exceeds $100°\mathrm{F}$, an *instance* of this event is produced. Each event (PE or CE) is associated with a unique identifier, called the *Event-ID*. A PE with ID i is represented as pe_i. Every instance of a particular event is also associated with a

unique ID, called the *Instance-ID*. The j^{th} instance of pe_i is represented as pe_i^j. Each event has a distinct schema to which every instance of that event adheres to. In addition to the Event-ID and Instance-ID, the event schema contains three mandatory attributes: *Node-ID*, which is the ID of the node at which the event originated, *Start-Time*, which is the time at which the particular event instance began (i.e., the time instance at which the event conditions were met), and *End-Time*, which is the time at which the event instance ended (i.e., event conditions ceased to be met). For instantaneous events (e.g., an RFID tag being scanned at a particular sensor), the Start-Time and the End-Time are both set to the same value. 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.

Similar to centralized CED schemes [20, 45, 46], our system supports a standard set of event composition operators (shown below). CEs are defined in terms of these operators and each operator incorporates a set of at least 2 PE arguments. 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. In other words, detecting ce_i requires monitoring of pe_k. 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 [20].

- **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 a pre-specified order. In other words, not only should at least one instance of every constituent PE occur within the sliding window, but they should 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. The **and**$(pe_1, pe_2, \ldots, !pe_i, \ldots, pe_m; w)$ operator specifies that an instance of CE is detected when at least one instance of each pe_1, pe_2, ..., pe_{i-1}, pe_{i+1}, ..., pe_m occurs within a window of length w and *no instance of* pe_i *occurs within the same time window*. The **seq**$(pe_1, pe_2, \ldots, !pe_i, \ldots, pe_m; w)$ operator specifies that an instance of CE is detected when at least one instance of each pe_1, pe_2, ..., pe_{i-1}, pe_{i+1}, ..., pe_m occurs *in the same order* within a window of length w and *no instance of* pe_i *occurs between* pe_{i-1}*.Start-Time and* pe_{i+1}*.End-Time*.

2.3.1 Problem Statement

Consider a set of m PE sources. Each PE source resides on one of the DTN nodes, although a DTN node may host more than one PE source. 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. In addition to EAP, EDP LT and BW, 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 dissertation, we regard the cost factor as a generic parameter. 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. However, the application can provide an alternate specification; for example, in case the nodes of the DTN are power constrained, the cost factor can be defined as the power consumed for transferring a packet over the link.

CEs are composed from the PEs using the above operators. For each CE, a node of the DTN is designated as its *destination* (also called as sink). 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). The destination node of ce_i is represented as $V_D(ce_i)$. The

user who defines a CE also specifies the maximum detection delay that can be tolerated for that CE. We call this the *delay tolerance limit* or simply the *delay tolerance* (represented as $\Delta(ce_i)$). The delay for a CE instance is the difference between the time at which the last constituent PE 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 of the various PE sources, (3) the frequency of each PE, and (4) the definition of the CE and its delay tolerance limit, the problem is to come up with a plan that minimizes the cumulative cost of detecting the CE. 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.

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 proactively 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.

2.3.2 CHALLENGES

With an example, we now explain two existing CED techniques and discuss why they cannot be trivially adopted for DTN settings. Figure 2.1(a) shows a scheduled contacts DTN with 7 nodes. There are four PEs $\{pe_1, pe_2, pe_3, pe_4\}$. which reside on nodes V_4, V_5, V_6 and 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.

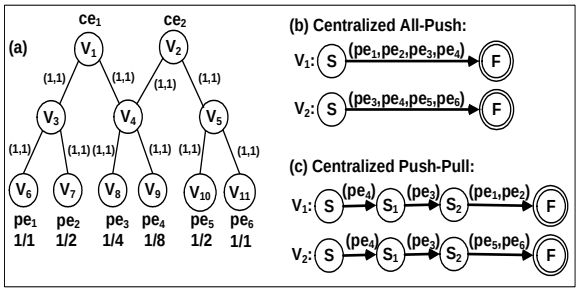


Figure 2.1: Centralized CED on DTN

The most straightforward approach to CED is to push every instance of each PE as and when they occur to the destination. The destination uses a sliding window to check which set of PE instances result in a CE instance. This approach is an adaptation of the centralized CED technique used in active databases and triggers [45, 46]. Figure 2.1(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 latency 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, on average one instance of pe_1 is produced every second. But, most of these instances 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 set to 2. The problem is that even if the user specifies a higher delay tolerance limit, this scheme fails to utilize it to lower the CED costs.

An alternate technique proposed by Akdere et al. [20], attempts to alleviate the problem by selectively pushing certain (usually the cheapest) PEs to the destination. The destination pulls the other PEs when it notices (based on PE instances that have arrived) that there is likelihood of a CE instance. The problem is to decide which PE sources will be pulled by the 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 prove that finding an optimal plan requires exponential time, and the paper proposes a heuristic algorithm. Their algorithm employs two kinds

of pulls. *single-target pulls* in which the node send 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.

However, note that even with this technique, the CED process is essentially executed at the destination, which means that again there is only one FSM in the plan. The plan consists 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 2.1(c) illustrates one such plan. Note that state S_1 corresponds to a single target pull, whereas S_2 corresponds to a multi-target pull. Even this technique suffers from several limitations. First, centralized CED can still result in significant degree of wasted communication. In Figure 2.1(a), one of the lowest-cost centralized CED plans 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 wasted communication and higher costs, especially if the cost factor of L_{13} is high. In general, it is better to discard such instances early (at nodes closer to the PE source). Second, due to frequent and potentially long link disconnections, pulling events by the destination would add significantly to the detection delay. The additional delay can be as much as twice the sum of the EDPs of the links forming the path from the destination to the PE source, because of the request/reply format of pulls. 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 structure is request/reply, and there can be a disconnection period before each.) 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 with a lower EDP; then, 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, thereby resulting in substantially higher CED costs and delays.

CHAPTER 3

EVENT AGGREGATION AND REDUNDANCY ELIMINATION[1]

[1] J. Chen, L. Ramaswamy, and D. Lowenthal. Towards efficient event aggregation in a decentralized publish-subscribe system. in Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, ser. DEBS '09. New York, NY, USA: ACM, 2009, pp. 18:1-18:11. Reprinted here with permission of publisher.

In this chapter, we present *Agele*, an event monitoring service that embodies our ideas towards reducing superfluous messages from potentially distinct event publishers in semi-real time.

3.1 Abstract

Recently, decentralized publish-subscribe (pub-sub) systems have gained popularity as a scalable asynchronous messaging paradigm over wide-area networks. Most existing pub-sub systems, however, have been designed with the implicit assumption that published data is clean and accurate. As the pub-sub paradigm is incorporated in real-world applications with human participants, this assumption becomes increasingly invalid due to the inherent noise in the event stream. The noise can take many forms, including redundant, in- complete, inaccurate, and even malicious event messages. This chapter explores the distributed computing issues involved in handling event streams with redundant and incomplete messages. Given a distributed broker overlay-based pub-sub system, we present our initial ideas for (1) aggregating event information scattered across multiple messages generated by different publishers and (2) eliminating redundant event messages. Key to our approach is the concept of an event-gatherer, a designated broker in the routing graph that acts as a proxy sink for all messages of a particular event, located at the graph center of the corresponding routing tree. This chapter proposes a novel decentralized algorithm to find this graph center. Results show that the proposed scheme typ- ically reduces the message load by over 60time overhead to subscribers.

3.2 System Overview

In this section, we provide an architectural overview of our system. Our event aggregation and redundancy elimination model is generic, and it can be applied to most pub-sub frameworks. However, for conceptual clarity, in this dissertation, we focus on a framework that is similar to type-and attribute-based pub-sub paradigm [7] (details about the subscription model is provided later in the section). However, the proposed architecture as well as the associated techniques can be adapted to topic-based or content-based publish-subscribe systems.

Agele is based upon a distributed overlay of message brokers (also referred to as *nodes*). The N message brokers in the overlay are represented as $\{b_1, b_2, \ldots, b_N\}$. Each broker is logically connected to a few other brokers such that the network forms a connected graph. Set of publishers and set of subscribers are represented as $\{p_1, p_2, \ldots, p_G\}$ and $\{s_1, s_2, \ldots, s_H\}$ respectively. Each publisher and subscriber must be connected to one of the brokers.

3.2.1 EVENTS, MESSAGES AND SUBSCRIPTIONS

Similar to type-and attribute-based pub-sub paradigm [7], every event in our system is associated with a *topic*, which provides a broad context for the event. Continuing with the example presented in the previous section, a traffic incident in a certain geographical area would represent a topic. In addition, events have a set of attributes (fields) that provide details of the event. The fields of an event e_q are represented as $\{e_q(1), e_q(2), \ldots, e_q(V)\}$. One of these fields (without loss of generality, the first field) is designated as the *event key*. Within a certain time-window, the key along with the topic corresponds uniquely (or can be resolved uniquely) to an event. In our system, the key field is descriptive, and it can be used in subscription predicates. For example, the key for a traffic incident event would be the street intersection at which it occurs. The number of fields of an event, their types and the key are determined by the event's topic.

However, in contrast to existing pub-sub systems, there can be multiple messages associated with a single event, and these messages may have been published by multiple publishers. The messages corresponding to an event e_q are represented as $\{e_q^1, e_q^2, \ldots,$
$e_q^U\}$. Each message provides some (possibly partial) information about the event. The fields of an event message e_q^r are represented as $\{e_q^r(1), e_q^r(2), \ldots, e_q^r(V)\}$. In a message that carries partial information about the event, one or more fields (other than the event key) would be empty. Our assumption of key-topic uniqueness implies that if the first message of an event is published at time t_f, any messages with the same key-topic pair generated between t_f and $t_f + W$ correspond to the same event.

In *Agele*, subscriptions are specified with respect to the event topic as well as its fields. A subscription has to necessarily identify the topic of interest. Additionally. it *may* specify predicates involving the fields associated with the topic. *Advertisements* are messages used by publishers to indicate the types of events they are going to generate. However, the system can be configured to work without advertisements, in which case it is assumed that every publisher can publish all types of events. Notice that the basic *Agele* framework can be considered as a special kind of content-based pub-sub model, where the content of every published item includes a topic and an event key. In fact, the *Agele* implementation is based on a content-based pub-sub system, Siena [3].

3.2.2 ROUTING

As in many pub-sub systems [2, 3], an acyclic network of brokers, called a *routing acyclic graph (routing AG, for short)* forms the basis for routing events from publishers to consumers. Routing AGs are embedded in the broker overlay. Notice that routing AGs are connected, non-directed, and acyclic (may also be considered as non-rooted trees). Similar to *Siena* [3] and *Hermes* [7], routing AGs in *Agele* are constructed in a completely decentralized fashion by peer-to-peer forwarding of subscriptions and advertisements. Furthermore, the predicates of subscriptions with the *same* topic are aggregated at brokers, and a more generic subscription is forwarded. In effect, *Agele* maintains a distinct routing AG for each topic. However, individual brokers can belong to multiple routing AGs. When available, advertisements can be utilized for optimizing the routing AGs.

3.2.3 EVENT GATHERERS

We now explain our mechanisms for message aggregation and redundancy elimination. Our strategy is to merge partial messages and eliminate redundant messages within the routing AG as they travel from their publishers to the subscribers. One of the nodes in the routing AG is designated as the *event-gatherer* for events disseminated through that routing AG. All the event messages with the same topic value are routed through the corresponding event gatherer. The event gatherer, upon receiving an event message, determines whether (1) this is the first message in a

possible sequence of messages from the same event, in which case it is stored, (2) it is redundant, and hence needs to be eliminated, or (3) it can be merged with an existing message. At an appropriate point in time (see below), the event-gatherer disseminates the aggregated message to the respective subscribers.

Concretely, an event-gatherer maintains a message buffer, which stores at most one message per distinct event. When a message comes in, the event gatherer checks whether an event message with the same event key already exists in the buffer. If so, there is an earlier message in the buffer corresponding to the same event. The event gatherer now determines if the newly arrived message is redundant (data contained in it is a subset of the data already available message existing in the buffer) or whether it can be merged with the existing message. Redundant messages are discarded. A message is *mergeable* if it contains some extra information that the existing message does not have (i.e., there exists one field which is empty in the existing message, but the corresponding field in the incoming message contains data). If the incoming message is deemed mergeable, the event gatherer creates a new aggregate message by combining the information available in the two messages and stores it in the buffer instead of the current buffered message. If the buffer does not contain an event with a matching key, the incoming message is inserted into the buffer.

An event gatherer is allowed to maintain the message corresponding to an event for at most W time units, since beyond this time another distinct event with the same event key might occur From message aggregation and redundancy elimination perspectives, it is optimal to have the event gatherers maintain the message corresponding for W time units, and after that send out the merged message to the actual recipients. However, waiting this long may sometimes affect the subscribers' ability to react properly to the event. To address this concern, the event gatherers periodically send out merged messages, while maintaining them in its buffer[2]. Specifically, for each event message in its buffer, the event gatherer evaluates the respective subscription predicates every T_m time units and sends out the merged message to a subscriber if its predicate is satisfied. However, subscription predicates might depend upon fields of the message that are currently *empty*. This issue can be

[2]Depending upon its buffer-space constraints, an event gatherer may store messages for shorter durations than what is permissible.

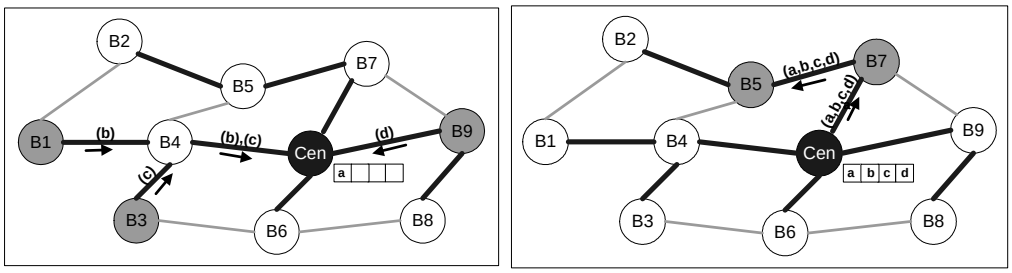


Figure 3.1: Pictorial representation of a topology with the center node indicated. All messages are part of the same event.

resolved in two ways - the event gatherer might simply wait until all the fields necessary for evaluating the respective predicate are available, or it might *optimistically* treat the predicate terms for which the values are not available as *true*. In the later case, there is a chance of *false notifications*.

Figure 3.1 provides an example of the idea of eliminating messages with an event gatherer (labeled *Cen* in the graph). On the left, the center node buffers a message with field *a*, while messages containing fields *b*, *c*, and *d* are traveling through the graph. On the right, the latter three fields pass through the center node and are merged into one message. Moreover, the new, merged message containing all four fields is buffered in case redundant messages later arrive at the center node.

3.2.4 Where Should an Event Gatherer be Located?

Next, we explore the crucial question of *where an event gatherer should be located?* In other words, *on which brokers of the routing AG should the event gathering functionality be placed?* The location of the event gatherer within the routing AG has a significant impact on the message load in the broker overlay. Our main objective when choosing the event gatherer location is to minimize the message load in the broker overlay. A second, but equally important, objective is to minimize the latency overheads on the subscribers.

We now formulate the problem in terms of the first objective. However, as explained in our technical report [47], our solution achieves both objectives. In the routing AG consisting of brokers $\{b_1, b_2, \ldots, b_M\}$, suppose the broker b_i is designated as the event gatherer. Now, if an event-message were to be generated by a publisher attached to node b_j, the messaging cost of transmitting this message to the event gatherer b_i is $Dist(b_j, b_i)$, where $Dist(b_j, b_i)$ represents the path length between the brokers b_j and b_i in terms of number of hops. Similarly, the cost of sending a merged message from the center to a subscriber b_k is $Dist(b_k, b_i)$. Since the event can enter into the system through any broker, the expected message load due to an event message is

$ECost(Rag, b_i) = \frac{1}{M} \sum_{b_j \in Rag} Dist(b_j, b_i)$,

where Rag represents the routing acyclic graph. Thus, the problem is to find a broker b_i such that $\sum_{b_j \in Rag} Dist(b_j, b_i)$ is minimized.

Traditional optimization strategies cannot be applied here as most of them are *centralized* schemes requiring a global view of the routing AG topology, whereas our system is based on a completely decentralized architecture wherein the brokers only interact with their neighbors. Furthermore, these strategies are computationally expensive. Therefore, we seek to develop an alternate solution that is not only amenable to decentralized implementation but is also efficient.

Our strategy is dependent upon the following observation. Since the cost of transferring a message from an arbitrary node b_j to the event gatherer b_i is determined by $Dist(b_j, b_i)$, in order to achieve low message loads, the event-gatherer should be located not too far away from any node in the routing AG. So, which broker in the routing AG satisfies this criterion? Intuitively, the event gatherer should be located in the *core region* of the routing AG. In fact, it can shown that the broker that satisfies this property would be the *graph center* of the routing AG. The graph center is defined as follows. Consider a undirected connected graph $G = (B, E)$ with nodes $B = \{b_1, b_2, \ldots, b_n\}$. The *distance* between two nodes b_j and b_i, represented as $Dist(b_j, b_i)$ is the length of the shortest path between them. The *eccentricity* of a node b_i is defined as the largest of the shortest paths between b_i and any other node in the graph. That is $Eccentricity(b_i) = Max_{b_j \in B}(Dist(b_i, b_j))$.

The graph centers of G are the set of nodes with minimum eccentricity [3]. Since, graph center has the minimal longest path, it naturally satisfies the above criterion.

3.3 GRAPH CENTERS

The problem now is to develop a completely decentralized algorithm to determine the center of acyclic graph, which would execute only through message exchanges among neighboring brokers. Several algorithms have been proposed for finding graph centers [48, 49]. However, very few of them are applicable to a decentralized setting where a global view of the network is not available. To the best of our knowledge there are very few distributed algorithms for locating centers of general graphs [50, 51]. However, even these algorithms have significant limitations, which prevents us from utilizing them.

Before presenting our solution, we define a few concepts that are necessary for its description. Consider two neighboring brokers b_i and b_j in the routing AG. Since the routing graph is acyclic (and undirected) , if the edge (b_i, b_j) is removed, the graph is partitioned. The partition (subgraph) that contains the node b_j is called the *subgraph anchored by* (b_i, b_j) and is represented as $SG(b_i, b_j)$. Similarly, the partition that contains b_i is the *subgraph anchored by* (b_j, b_i) ($SG(b_j, b_i)$). Note that in the routing AG, the path from b_i to any node in $SG(b_i, b_j)$ has to necessarily pass through b_j. The subgraph eccentricity of b_i with respect to b_j (represented as $SEcc(b_i, b_j)$) is defined as the eccentricity of b_i with respect to only the nodes in $SG(b_i, b_j)$. In other words, $SEcc(b_i, b_j) = Max_{b_x \in SG(b_i,b_j)}(Dist(b_i, b_x))$. The neighboring brokers of b_i in the routing AG is represented as $NbrList(b_i)$.

3.3.1 DISTRIBUTED CENTER DETERMINATION

As we remarked earlier, our algorithm is completely decentralized, and it relies purely on message exchanges among neighbor brokers. The algorithm works in three stages, as explained below.

[3] There could be multiple graph centers, all of which have minimum eccentricities

INITIATION PHASE

In this phase any broker in the routing AG initiates the center determination algorithm. It does so by sending an initiation (INIT) message to all its neighbors in the routing AG. The INIT message contains the routing AG topic and the identifier of the broker initiating the process. Each broker receiving the INIT message propagates it to all its neighbors in the routing AG. Concurrent initiations are resolved by ignoring all but the first message.

ECCENTRICITY DETERMINATION PHASE

In the second phase, the brokers in the routing AG progressively discover their eccentricities with respect to the routing AG. A broker b_j sends one or more messages to its neighbor broker b_i, each of which updates the current value of $SEcc(b_i, b_j)$. Finally, when b_j discovers that current value of $SEcc(b_i, b_j)$ has reached its final value, it sends a STABLE message to b_i.

Concretely, the algorithm works as follows. Suppose b_i and b_j are neighboring brokers in the routing AG, and suppose $NbrList(b_i)$
$= \{b_e, \ldots, b_h, b_j\}$ and $NbrList(b_j) = \{b_i, b_k, \ldots, b_p\}$. Every broker in the routing AG maintains its subgraph eccentricity with respect to each of its neighbors. Further, it also maintains the subgraph eccentricities of each of its neighbors with respect to itself. For example, broker b_i maintains $SEcc(b_i, b_x)$ and $SEcc(b_x, b_i)$ $\forall b_x \in NbrList(b_i)$. When a broker receives an initiation message, it initializes its subgraph eccentricities with respect to each of its neighbors to zero.

When a leaf broker, denoted b_k, receives an INIT message from its only neighbor (b_j), it sends back a UPDATE message containing $SEcc(b_j, b_k)$, which is equal to 1 (as b_k is the only node in the subgraph reachable through itself). It also updates its local copy of $SEcc(b_j, b_k)$ to 1. Further, b_k sends a STABLE message to b_j indicating that $SEcc(b_j, b_k)$ has stabilized. When an intermediate node b_j receives an UPDATE message from its neighbor b_k, it updates its copy of $SEcc(b_j, b_k)$. Then, for each of its neighbor brokers, b_j checks whether its subgraph eccentricity with respect to b_j needs to be updated. $SEcc(b_i, b_j)$ needs to be updated if the current value of $SEcc(b_i, b_j)$ is less

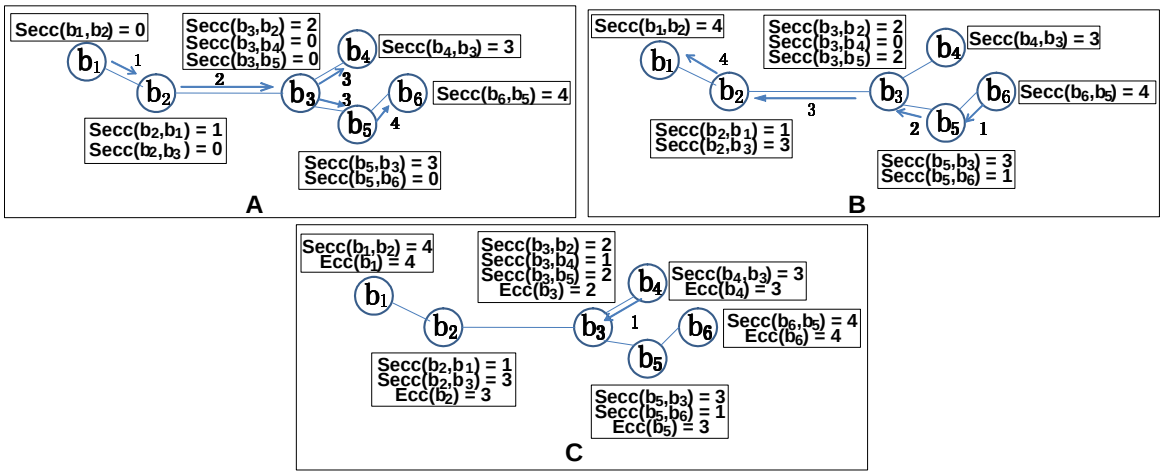


Figure 3.2: Pictorial representation of the eccentricity determination phase.

than $SEcc(b_j, b_k) + 1$. If $SEcc(b_i, b_j)$ needs to be updated, b_j sets its local copy of $SEcc(b_i, b_j)$ to $(SEcc(b_j, b_k) + 1)$, and sends the same value as an UPDATE message to b_i.

Notice that $SEcc(b_i, b_j)$ can only updated when b_j receives an UPDATE message from one of its neighbors other than b_i. Therefore, if b_j receives a STABLE message from all of its neighbors other than b_i, it can safely conclude that $SEcc(b_i, b_j)$ will not be updated further. Thus, when b_j receives a STABLE message from all of its neighbors other than b_i, it sends a STABLE message to b_i.

When b_i receives STABLE messages from *all* its neighbors, it decides that its subgraph eccentricity values with respect to all of its neighbors has stabilized. It can now compute its eccentricity value as the maximum of its subgraph eccentricity values with respect to each of its neighbors. Formally,

$Ecc(b_i) = Max_{(b_x \in NbrList(b_i))}(SEcc(b_i, b_x))$.

Figure 3.2 illustrates the eccentricity determination phase on a sample routing AG consisting of 6 brokers. It is assumed that all the brokers have received the INIT messages, and the leaf brokers are about to start the process of sending UPDATE messages. All $SEcc$ values at each broker are initialized to zero. For better comprehensibility, the figure shows the three leaf brokers sending

out their UPDATE messages *sequentially*. However, it is important to note that the algorithm itself does not place any such sequentiality restrictions. Further, for simplicity, we do not show STABLE messages. Figure 3.2-A shows the propagation of UPDATE message from b_1. The figure also shows the $SEcc$ values of every broker with respect to all of its neighbors. These are values resulting from the propagation of b_1's UPDATE message. Similarly, Figure 3.2-B shows the dissemination of an UPDATE message from b_6 and the various $SEcc$ values resulting from this dissemination. Notice that b_6's message is not propagated to b_4, because $SEcc(b_4, b_3)$ is already 3, and it would not be altered by b_6's message. Similarly, in Figure 3.2-C b_4's UPDATE message is not propagated beyond b_3. Figure 3.2-C also shows the eccentricity values of all the brokers in the routing AG.

CENTER DETERMINATION PHASE

Once a node discovers its eccentricity, it sends its eccentricity value to all of its neighbors. A broker whose eccentricity is less than or equal to the eccentricities of all of its neighbors determines that it is one among possibly many centers of the routing AG. Upon discovering that it is a center of the routing AG, the broker sends an announcement message to all its neighbors, which is then propagated through the routing AG. Upon completion of this phase, every broker in the routing AG knows about all the centers of the routing AG. Note that a routing AG might have more than one center. However, if it has multiple centers, these centers form a connected subgraph (i.e., any broker that lies in between two center brokers should itself be a center). The center broker is anointed as the event gatherer for the routing AG. If there are multiple centers, ties may be broken in favor of the broker with the smallest id. Notice that in Figure 3.2-C, b_3, having the least eccentricity, discovers itself as the center of the routing AG.

3.3.2 PROOF AND ANALYSIS OF ALGORITHM

We outline the termination and correctness proofs of our distributed algorithm to find the center of a routing AG. Furthermore, we theoretically analyze its messaging costs.

TERMINATION

Recall that each broker sends a single INIT message to its neighbors in the first phase. Similarly, in the center determination phase, brokers send a single message to their neighbors to communicate their respective eccentricities. Further, the brokers also send a single message to their neighbors for disseminating the center information. Thus, we just need to show that the eccentricity determination phase of the algorithm always terminates, which we do by induction.

Recall that the eccentricity determination phase concludes when every broker receives a STABLE message for each of its neighboring brokers. Without loss of generality consider two neighboring brokers b_i and b_j. We will now show that b_i receives a STABLE message from b_j within a finite time. We distinguish between two cases, namely b_j being a leaf broker and b_j being an intermediate broker. If b_j is a leaf broker (the base case), it sends a STABLE to b_i message as soon as it sends the only UPDATE message to b_i.

In case b_j is not a leaf node, we use induction. The routing AG can be divided into two subgraphs $-$ a subgraph anchored by (b_i, b_j), (denoted $SG(b_i, b_j)$) and a subgraph anchored through (b_j, b_i) (denoted $SG(b_j, b_i)$). Recall that the $SG(b_i, b_j)$ consists of nodes that are reachable from b_i by only passing only through b_j. In our algorithm, b_j sends a STABLE message to b_i as soon as it receives a STABLE message from all of its neighbors other than b_i. Notice that except for b_i, all other neighbors of b_j lie in $SG(b_i, b_j)$. Thus, the act of b_j sending a STABLE message to b_i can recursively depend upon the state of the brokers in $SG(b_i, b_j)$, but it can never depend upon any broker in $SG(b_j, b_i)$. The same argument can now be inductively applied for each neighbor of b_j and the subgraphs anchored by the corresponding edges. Observe that with each successive application of the logic we are dealing with a smaller subgraph. Since the routing AG has finite number of nodes, eventually the subgraphs contain only leaf nodes, which is covered by the base case. Hence, within a finite number of messages, b_j will send a STABLE message to b_i.

ALGORITHM CORRECTNESS

We prove the correctness of our algorithm in two steps. First, we show that our algorithm ensures that every broker in the routing AG correctly discovers its eccentricity. Then, we outline the correctness argument for our center determination protocol (involving eccentricity value exchanges among neighboring nodes). Recall that if b_i and b_j are neighboring brokers $SG(b_j, b_i)$ denotes the subgraph anchored by (b_j, b_i) and $SEcc(b_i, b_j)$ represents the subgraph eccentricity of b_i with respect to b_j. $Ecc(b_i)$ represents the eccentricity of b_i, $NbrsList(b_i)$ denotes the neighbor of b_i in the routing AG, and $Dist(b_i, b_k)$ represents the distance between b_i and b_k.

The correctness argument for eccentricity determination phase itself involves two steps. In the first, we show that in an undirected acyclic graph, the eccentricity of any node can be correctly computed as the maximum of its subgraph eccentricities with respect to all of its neighbors. In the second step we demonstrate that through the proposed algorithm, any arbitrary node in the routing AG can correctly determine its subgraph eccentricity with respect to every neighbor. Consider an arbitrary node b_i in the routing AG. Let b_k be the most distant node from b_i in the routing AG (thus the distance between b_i and b_k is b_i's eccentricity). Suppose b_k lies in $SG(b_i, b_j)$, then $SEcc(b_i, b_j) = Dist(b_i, b_k)$. The acyclic nature of the routing graph implies that there is exactly one path from b_i to b_k in the routing path, and that path entirely lies in the subgraph $SG(b_i, b_j) \cup b_i$. Thus, the distance between b_i and b_k in subgraph $SG(b_i, b_j) \cup b_i$ is the same as the distance between them in the original routing AG. The acyclic nature also implies there is also no other node in $SG(b_i, b_j)$ that is at a greater distance from b_i than b_k. Further, the acyclic property can be used to show that $SEcc(b_i, b_x) \leq Dist(b_i, b_k); \forall b_x \in NbrList(b_i)$. Thus, $Max_{b_x \in NbrList(b_i)}(SEcc(b_i, b_x)) = Dist(b_i, b_k) = Ecc(b_i)$.

Now, we demonstrate that our algorithm ensures that every broker correctly discovers all of its $SEcc$ values. Again, consider two neighboring brokers b_i and b_j. Observe that if b_j forwards an UPDATE message originating at an arbitrary leaf node (say b_l) to b_i, then that value of that

message would be $Dist(b_i, b_l)$. Next, if b_k is the most distant node in $SG(b_i, b_j)$[4], then the UPDATE message from b_k would be eventually forwarded by b_j to b_i. The only scenario in which b_j does not forward the message from b_k would be when it has already received a message with a higher value from another node in $SG(b_i, b_j)$. However, this cannot happen since b_k is the most distant node in $SG(b_i, b_j)$. Thus, b_i correctly discovers $SEcc(b_i, b_j)$.

Next, we outline the correctness proof for our center determination protocol. Recall that in the third phase each broker compares its eccentricity information with all of its neighbors, and a node b_i discovers that it is the center if the satisfies the twin conditions ($Ecc(b_i) \leq Ecc(b_x); \forall b_x \in NbrsList(b_i)$ and $\exists b_y \in NbrsList(b_i)$ such that $Ecc(b_i) < Ecc(b_y)$). The correctness proof is based on the following three lemmas, whose proofs we omit due to space limitations. (1) In an acyclic graph, at least one neighbor of a center node has higher eccentricity that the center node. (2) For two arbitrary non-neighbor nodes b_i and b_j of a connected acyclic graph, if b_x is a node in the path between them, then it satisfies the condition $Ecc(b_x) \leq Max(Ecc(b_i), Ecc(b_j))$. (3) In an acyclic graph, if the node b_k is the most distant node to b_i, then the path from b_i to b_k always passes through at least one graph center. Using these three lemmas we can show that for node eccentricities of a connected acyclic graph, the local minima is also the global minimum. This implies that the center nodes and only the center nodes satisfy the two conditions.

ALGORITHM ANALYSIS

Next, we analyze the message costs of our decentralized center determination algorithm. The three phases of the algorithm are considered separately, and for each phase we try to determine an upper bound for the number of messages circulated in the routing AG.

In the initiation phase, the initiation message sent by one of the nodes is sent to all brokers. If there are M brokers in the routing AG, the total number of messages circulated in this phase is $(M - 1)$. Thus, the message load for this phase is $O(M)$.

[4]For simplicity, we assume that there is only one most distant node in $SG(b_i, b_j)$. However, this assumption can be easily relaxed.

Analyzing the message load in the eccentricity determination phase is a bit tricky. Unlike the first phase, we cannot determine the exact number of messages circulated in the routing AG during this phase. In this phase each leaf broker sends out an UPDATE message, which is then circulated to some brokers in the routing AG. The extent to which an UPDATE message is circulated in the routing AG depends upon the subgraph eccentricity values of various brokers when the message reaches them, which in turn depends upon other UPDATE messages that have traversed through that node. Thus, the exact number of UPDATE messages is non-deterministic. Even if every UPDATE message reaches every single node of the routing AG, the total number of messages circulated during this phase would be no more than $L \times (M - 1)$, where L represents the number of leaves in the routing AG. Thus, $L \times (M - 1)$ is an upper bound on the message load for the second phase. However, the actual number of messages circulated during this phase is much smaller.

The third phase involves two distinct operations, namely brokers communicating their eccentricity values to their neighbors and disseminating center information to all brokers in the routing AG. The first operation results in $2 \times (M - 1)$ messages and the second operation results in $(M - 1)$ messages. Therefore the total message load for this phase is $3 \times (M - 1)$. Thus, the message load of the entire algorithm is no more than $(L + 4) \times (M - 1)$, but typically is much smaller.

3.4 AGELE IMPLEMENTATION

This section describes the *Agele* implementation. *Agele* is built on top of the *Siena* simulator [3]. It implements the graph center determination algorithm discussed in Section 3.3. It also eliminates redundant and partial messages. For completeness, we first describe briefly the *Siena* system. Then, we describe *Agele*.

3.4.1 SIENA

Siena (Scalable Internet Event Notification Architectures) is a wide-area event-based notification system [3]. It was developed at the University of Colorado. In *Siena*, a message consists of a

set of typed attributes. Each attribute has a unique name, type and value. A predicate consists of the disjunction of conjunctions of elementary constraints, where each element constraint is a quadruple (name, type, operator, value). We also adapt the combined broadcast and content-based (CBCB) routing scheme [52] for a content based network. In CBCB routing, the broadcast tree of each message is pruned so that the message is only propagated to those nodes that issued the matching predicates. The routing information in the content based network is propagated in the network using push and pull mechanisms. In the push mechanism, the subscribers push the Receiver Advertisement (RA) to the potential publishers. This sets the routing table along the path, which allows the messages to be forwarded to the subscribers. In the pull mechanism, Sender Requests (SRs) and Update Replies (URs) are applied to maintain the routing table. The router sends the SRs to pull a content based address from other routers, and the routers that are queried reply with URs that contain updated content based address.

3.4.2 AGELE

We added significant infrastructure to *Siena* to create *Agele*. First, we added the center-finding algorithm given in the previous section. This algorithm is run at the start of the simulation, concurrently while the simulation is proceeding. In other words, eliminating redundant and partial messages only occurs after the graph center is found. (However, the center is found quickly, so this effect is minor.)

To implement the *Agele* center-finding algorithm, we construct a connected acyclic overlay for each topic on top of the network of brokers. The algorithm runs on the overlay to find the graph center. All control messages related to the establishment of the graph center are routed through the overlay. Upon the termination of graph center algorithm, each node will have a forwarding table that contains the information to force messages originated from publishers to be routed through the graph center. Once the messages are processed at the graph center, the processed messages are propagated to the subscriber using the content based network routing table [52].

Second, we implemented elimination of redundant and partial messages. To do this, we added a buffer at the center node. When a message M_a arrives at the center, it is buffered. The idea is that a message will be buffered for a period of time, during which incoming messages will be compared for redundancy; however, incoming messages will be considered for merging for no more than (but possibly less than) the buffering time. Two different timers are kept: one is set to T_r, the redundant threshold, and the other to T_m, the merge threshold. The redundant threshold must always be at least as large as the merge threshold (see below).

If a message M_b arrives before the redundant timer expires, and M_b is a subset of M_a, then message M_b is dropped. On the other hand, if M_b is not a subset, and it can be merged with M_a, then a message M_{ab} is created and buffered. In addition, M_a and M_b are removed from the buffer.

On the other hand, if no mergeable message arrives before the merge timer for M_a expires, then M_a is forwarded on from the center node to the next node (according to the forwarding table). However, note that M_a remains in the buffer of the center node. It is removed only when the redundant threshold is reached. As an optimization, if a complete message is ever received at the graph center, it is immediately forwarded, independent of either timer.

3.5 Experimental Results

This section reports the results obtained using *Agele*. We first describe the experimental setup and then move on to specific results.

3.5.1 Setup

Our experiments were set up as follows. Each complete event in our experiments consists of 20 fields including the event key. In published messages, the number of fields that holds valid data varies from 1 to 10. The number of messages pertaining to an individual event can vary, and they are generated in the following manner. Each publisher of a particular event generates messages pertaining to that event according to a Poisson process. The event duration (time window in which the messages of an event are generated), however, is chosen to be such that it falls within T_b, which

we define as the maximum amount of time that any event can take (our experiments investigate different values for T_b). Intuitively, real-world events will likely consist of a burst of messages in a relatively short time period. In our experiments, all nodes subscribe once and for any event, 20% of nodes publish messages pertaining to it. The particular event and associated field names are selected according to a uniform random distribution. We have experimented with two kinds of broker overlay topologies, namely random and power-law networks.

We varied separately the merge threshold, T_m, and the redundant threshold, T_r. Note that ideally, T_r would be set to W (the event time window), but it might not be known a priori, and even if it is, buffer capacities might mandate a smaller value. In our experiments, these ranged between 0 and 10 simulated time units, such that $T_m \leq T_r$. The buffer requirement at the center node was quite modest; it was never more than 100 messages, even on simulations of 3200 nodes.

If T_m and T_r are both 0, then the center node becomes merely a "pass-through", and subscribers will incur additional overhead with no benefit to the overall system. Nevertheless, this is useful from a measurement perspective, as it allows us to isolate the overhead for re-routing messages through the center node.

Generally, we examine the two key metrics: what percentage of the messages were suppressed and how much extra time was added due to using a center node and buffering messages. We use the term *suppressed* because a message can be eliminated (as a duplicate) or merged (with a like event); either way, that message does not emerge from the center node. Note that the minimum number of messages that a subscriber can receive is one. That is, if T_m and T_r are both sufficiently large (generally much larger than the values that we use), all messages will be held at the center node until just one message is forwarded on. For the latter, we measure time per event as the difference between the time that the event is fully received by the subscriber and the time that the first message of that event is published. As the event generation is random, all results presented are the median of several test runs (typically 50).

We start by presenting several results with a random graph and random message origination. All results (except where noted) are relative to *Siena*, where there is no notion of a center node

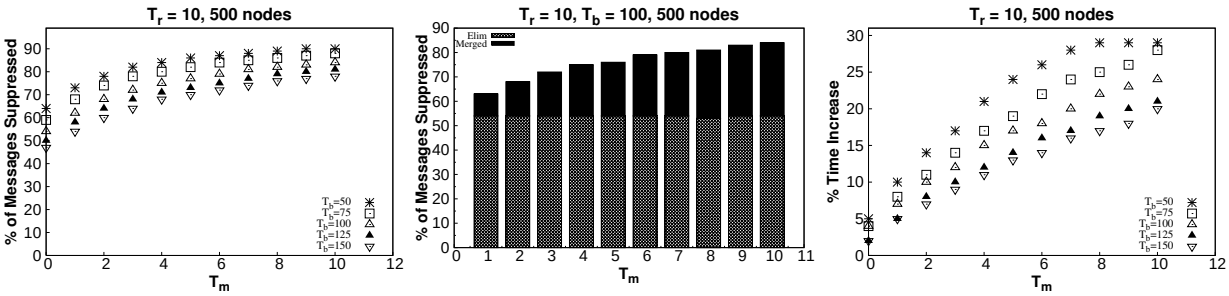
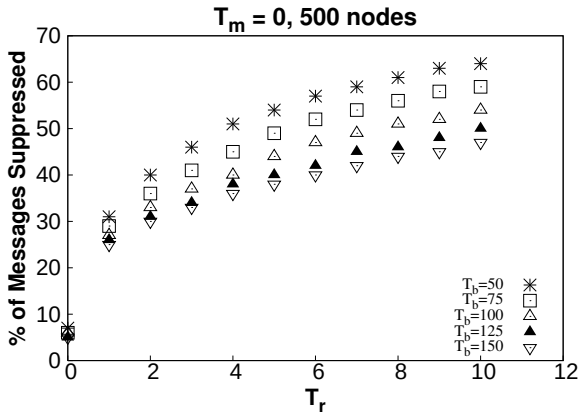


Figure 3.3: Percentage of messages suppressed, breakdown of whether the suppressed messages were duplicates or merged, and time increase when T_m varies.



T_b	% Time Increase
50	6
75	4
100	4
125	3
150	4

Figure 3.4: On the left, percentage of messages suppressed when T_r varies. On the right, for $T_r = 10$, the time increase for five different values of T_b.

and no buffering of messages. Note that in this section, messages suppressed refers to *originated messages* from a publisher. In addition, T_m and T_r are measured in simulated time units.

3.5.2 VARYING T_m AND T_r

First, we investigate the effect of varying T_m (see Figure 3.3) for five different values of T_b (50, 75, 100, 125 and 150 time units). For this experiment, we set T_r to 10 and use 500 nodes. The number of messages suppressed (left-hand graph) is only somewhat dependent on T_m—with the large redundant threshold, many messages are suppressed (especially when T_m is small) even though

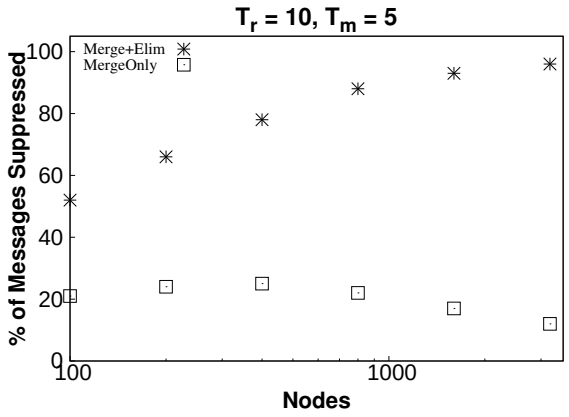
they conceptually could be merged. (In other words, as one might expect, dropping a message takes priority over merging one.) In addition, the percentage of messages that are suppressed is largest when T_b is smallest. This is expected, as when messages are clustered, there is a greater chance that when one message from a given event arrives at the graph center, other messages from the same event arrive before T_m time units. Note that the percentage of suppressed messages levels off in our range of T_m values because events with few messages cannot be suppressed in practice.

The right-hand graph shows the cost of suppressing messages, which ranges from a 5% to 34% overhead. When T_b is 100, the overhead is quite good—at most 24%. Furthermore, choosing T_m to be 4, we limit the overhead to 15% yet suppress over 70% of the messages. This would likely be a good tradeoff. In a real-world system, one benefit of suppressing—which we are not considering here, but will in future work—is that bandwidth consumption is lower. This in turn will decrease potential message drops due to overloaded nodes. Therefore, the time for a subscriber to receive an event will likely be *lower* when using Agele than baseline Siena.

Second, we investigate the effect of varying T_r (see Figure 3.4). For this experiment, we set T_m to 0. It is clear that the number of messages suppressed (left-hand graph) is strongly dependent on T_r. The effect is similar as the number of nodes increases. The time increase is independent of T_r, because time overhead occurs only when messages are buffered at the center node for potential merging. Therefore, we show the effect on time increase when varying T_b. The table on the right-hand side of the figure shows that the time increase, for a fixed value of T_r (10 in this case), is only slightly dependent on T_b (which is varied between 50 and 150).

3.5.3 SCALABILITY

We next investigate the scalability of *Agele* by varying the number of nodes from 100 to 3200, doubling the number each time (see Figure 3.5). For this experiment, we set T_m to 5 and T_r to 10. The left-hand graph contains two plots. One is the percentage of total messages suppressed, and one is only the number of messages merged. The total messages suppressed *increases* with the number of nodes. This is because there is a fixed number of events, and hence more duplicate messages



Number of Nodes	% Time Increase
100	17
200	32
400	27
800	29
1600	20
3200	22

Figure 3.5: Percentage of messages suppressed and time increase when the number of nodes varies. The graph uses a log scale for the x-axis.

Eccentricity	Normalized Event Time (s)
7	18
8	53
9	340
10	1263
11	1572

Nodes	Time to Find Graph Center (s)
100	10
200	11
400	15
800	16
1600	19
3200	20

Figure 3.6: On the left, event times for aggregator nodes using different eccentricities. On the right, the time to find the graph center for various node counts, normalized based on the number of aggregator nodes used.

are suppressed. Note that in a real-world system, increasing the number of human participants will likely only slightly, if at all, increase the number of events, assuming that the geographical area is fixed. The number of messages merged—which is more difficult to scale—remains relatively constant as the number of nodes increases. Not until 1600 nodes does this number start to fall off, and it does so slowly.

The right-hand table shows the time increase relative to Siena. The overhead is between 17% and 32%, which is quite good. Overall, *Agele* scales well especially considering that typical pub-sub systems contain less than 1000 broker nodes.

3.5.4 Graph Centers

The *Agele* system is predicated on finding the graph center, because this is the most effective node through which messages should be routed. To show the effectiveness of finding the center, we performed the following experiment. We started with a 300-node random graph, where the center node has an eccentricity of 7. We tried using as aggregators all nodes with eccentricity value 8. We repeated this (separately) using values 9, 10, and 11. The idea is that using centers with larger eccentricities will yield inferior results.

The left-hand side of Figure 3.6 shows the results. We measured the average time for an event in *Agele* over aggregators with five different eccentricities. Because there are more nodes with eccentricity 8 than 7, 9 than 8, etc., we normalize the results based on the number of centers. (This is because with more aggregators in general means lower event times.) The figure clearly shows that using aggregators with larger eccentricities produces inferior results. Hence, using center nodes to aggregate is critical.

Finally, the right-hand side of Figure 3.6 presents the time to find the graph center for different numbers of nodes (ranging from 100-3200). This shows clearly that the center finding algorithm within *Agele* is efficient.

CHAPTER 4

ENHANCED EVENT AGGREGATION AND REDUNDANCY ELIMINATION [1]

[1] Jianxia Chen; Ramaswamy, L.; Lowenthal, D.K.; Kalyanaraman, S.; , "CAEVA: A customizable and adaptive event aggregation framework for collaborative broker overlays," Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on , vol., no., pp.1-9, 9-12 Oct. 2010. Reprinted here with permission of publisher.

In this chapter, we describe the design, implementation, and performance of *Caeva* [53], which is a decentralized, dynamic, and configurable event monitoring system that handles redundant and partial events.

4.1 Abstract

The publish-subscribe (pub-sub) paradigm is maturing and integrating into community-oriented collaborative applications. Because of this, pub-sub systems are faced with an event stream that may potentially contain large numbers of redundant and partial messages. Most pub-sub systems view partial and redundant messages as unique, which wastes resources not only at routers, but also at possibly resource constrained subscribers. In this chapter, we present Caeva, a customizable and adaptive event aggregation framework. The design of Caeva exhibits three novel features. First, the tasks of merging messages and eliminating redundancies are shared among multiple, physically distributed brokers called aggregators. Second, we design a decentralized aggregator placement scheme that continuously adapts to decrease messaging overheads in the face of changing event publishing patterns. Third, we allow subscribers to choose a notification schedule that meets their specific needs. Results of extensive experiments show that Caeva is quite effective in providing flexibility and efficiency.

4.2 System Overview

Caeva is a collaborative, distributed-overlay based pub-sub infrastructure that supports event message aggregation and redundancy elimination in addition to routing messages from publishers to subscribers. Its design is motivated by *Agele* [44], which is described further in Chapter 3. In this section, we first describe the architecture of *Caeva*. Then, we discuss decentralized, adaptive aggregation. Finally, we discuss customizing a notification schedule at the subscriber.

4.2.1 ARCHITECTURE

Caeva is built upon a distributed overlay of message brokers (also referred to as *nodes*), represented as $\{b_1, b_2, \ldots, b_N\}$. Each broker is logically connected to a few other brokers such that the network forms a connected graph. The set of publishers and set of subscribers are represented as $\{p_1, p_2, \ldots, p_G\}$ and $\{s_1, s_2, \ldots, s_H\}$ respectively, with each publisher and subscriber connected to one of the brokers.

Caeva's subscription model is similar to type-and attribute-based pub-sub paradigm [7]. However, the proposed architecture as well as the associated techniques can be adapted to topic-based or content-based pub-sub systems. Every event in our system is associated with a *topic*, which provides a broad context for the event. For example, a traffic incident in a certain geographical area would represent a topic. In addition, events have a set of attributes (fields) that provide details of the event. The fields of an event e_q are represented as $\{e_q(1), e_q(2), \ldots, e_q(V)\}$. One of these fields (without loss of generality, the first field) is designated as the *event key*. The key field is descriptive, and it can be used in subscription predicates. For instance, the key for a traffic incident event would be the street intersection at which it occurs. Within a certain time-window, the key along with the topic corresponds uniquely to an event. The number of fields of an event, their types, and the key are determined by the event's topic. Subscriptions are specified with respect to the event topic as well as its fields. A subscription has to necessarily identify the topic of interest. Additionally, it *may* specify predicates involving the fields associated with the topic.

There can be multiple published messages associated with a single event (represented as $\{e_q^1, e_q^2, \ldots, e_q^U\}$ for event e_q) , possibly published by multiple publishers. Each message contains a subset of fields of the corresponding event. The fields of an event message e_q^r are represented as $\{e_q^r(1), e_q^r(2), \ldots, e_q^r(V)\}$. According to key-topic uniqueness assumption, if the first message of an event is published at time t_f, any messages with an identical key-topic pair generated between t_f and $t_f + W$ correspond to the same event. Publishers may *advertise* the types of events they are going to generate. However, the system can be configured to work without advertisements, in which case it is assumed that every publisher can publish all types of events.

Similar to many existing pub-sub systems [2, 3], *routing AGs graphs* comprised of brokers from the overlay form the basis for routing events from publishers to subscribers. Routing AGs are constructed in a completely decentralized fashion by peer-to-peer forwarding of subscriptions and advertisements. The predicates of subscriptions with the *same* topic are aggregated at brokers using the subsumption relationship, and a more generic subscription is forwarded. While *Caeva* maintains a distinct routing AG for each topic, individual brokers can belong to multiple routing AGs.

4.2.2 Decentralized, Adaptive Aggregation

Caeva uses a collaborative, decentralized and adaptive approach to aggregating events and eliminating redundancy. At a high-level, decentralized aggregation has a resemblance to the operator placement problem in distributed stream processing systems [54]. The question, therefore, is whether similar techniques can be used for the problem at hand. However, in a community-based event system, message publishers (source nodes) of a particular event are not known before hand, which precludes adopting heavyweight, plan-based techniques that have been used for distributed stream processing systems. We need a lightweight and dynamic scheme that does not need apriori knowledge of message sources of an individual event.

In our approach, designated brokers within the routing AG of a particular event type participate in aggregating and eliminating events of that type. Such brokers are referred to as *aggregators*. Each aggregator is autonomous and maintains a buffer that stores part of an event.

In *Caeva*, we coordinate the activities of the various aggregators of an event. This ensures that subscribers receive event information available in one composite message at the end of each notification cycle. A subset of aggregators, called *active aggregators* (AAs), additionally perform coordination. One of the active aggregators, the *coordinator*, coordinates the final round of aggregation and routes the aggregated message to subscribers. We denote all non-active aggregators as *passive aggregators* (PAs). *The key to* Caeva *is that the aggregators are chosen dynamically, and then are moved adaptively when necessary.*

In the next two subsections we explain the operations of active and passive aggregators and the coordinator. In turn, we discuss the dynamic aggregation within *Caeva*, its coordination algorithm for the active aggregator, and then how aggregators are placed within the broker overlay and moved adaptively.

In this discussion, we focus on the routing AG of a single event type. However, multiple routing AGs can simultaneously exist in *Caeva*, and the techniques and mechanisms discussed below apply to the routing AGs within the broker overlay. For now, we assume all subscribers have the same notification cycle duration; the next section relaxes this assumption.

Notationally, the set of passive aggregators is denoted $PvSet = \{pv_1, pv_2, \ldots, pv_F\}$ and its active aggregator set $AvSet = \{av_1, av_2, \ldots, av_G\}$. The coordinator of the event e_q is represented as C_q.

DYNAMIC AGGREGATION

When the event message e_q^r reaches a passive aggregator pv_f, there are three possible cases: **(1)** pv_f has a message corresponding to the event e_q in its buffer, and that message is a superset of all the fields contained in e_q^r. In this scenario, e_q^r is redundant and therefore dropped. **(2)** pv_f has a message pertaining to event e_q in its buffer, but that message does not have all the fields contained in e_q^r. In this case ,e_q^r is merged with the buffered message. **(3)** e_q^r is the first message of event e_q. Here, pv_f inserts it into its buffer, but also passes it to its upstream neighbor; it will eventually reach an active aggregator. PA pv_f will eventually get a reply back from the active aggregator indicating the coordinator and notification cycle. pv_f sends the (partially) aggregated message in its buffer to C_q just before the end of every notification cycle (the manner in which pv_f discovers C_q and the duration of e_q's notification cycle is discussed later).

An active aggregator (say av_g), upon receiving an event message e_q^r, behaves identically to a passive aggregator except in case 3. In that case, it first checks whether another active aggregator is already designated as the coordinator of e_q. If so, it just inserts e_q^r into its buffer as the first message of e_q. AA av_g will eventually finds out the notification cycle details from e_q's coordinator (if it

does not know already). If av_g is not aware of any other node claiming the coordinator-hood of e_q, it executes the coordinator establishment protocol described in the next sub-section. In all three scenarios, if e_q^r was sent to av_g by a passive aggregator, av_g informs the passive aggregator about the coordinator and the notification cycle details of e_q.

The coordinator performs all the aggregation-related duties described above. In addition, at the end of every notification cycle, it receives partially aggregated messages from passive and active aggregators. These messages are merged and any redundancies are eliminated. The merged message is then sent to the subscribers.

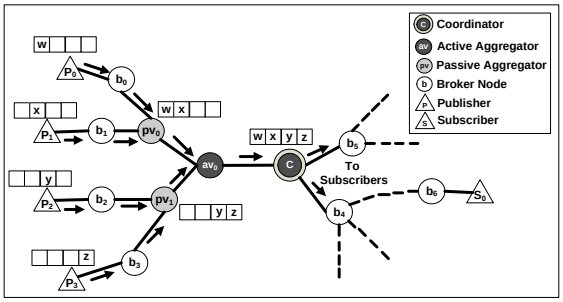


Figure 4.1: Distributed Message Aggregation in *Caeva*

Figure 4.1 depicts the multi-stage merging at the passive/active aggregators and the coordinator.

DYNAMIC COORDINATION

In *Caeva*, coordination involves two parts: **(1)** establishing the coordinator for an event and **(2)** informing the relevant set of aggregators of the coordinator identity and the notification cycle details. This way, other aggregators can forward aggregated messages to the coordinator at the appropriate time. Designing scalable and efficient coordination in loosely coupled systems such as *Caeva* is challenging. In order to limit the overheads, we confine most coordination-related duties to active aggregators.

When an active aggregator av_g receives a message of an event e_q with no established coordinator, av_g attempts to become the coordinator. It sends a message to all other active aggregators. An active aggregator av_h receiving such a message from av_g consents to av_g's claim if av_h has not

attempted to become the coordinator of e_q. Ties are broken in decreasing order of broker ID; the "winner" sends a denial message to the "loser", who consents.

Once the coordinator is established, it determines the duration of the notification cycle and the start time of the first cycle. With the assumption (for now) that all subscribers have the same fixed notification cycle duration, determining the cycle duration is trivial. The coordinator sends its identity and the notification cycle to the relevant set of aggregators; these aggregators in turn forward partially aggregated messages to the coordinator "just in time" (before the end of the notification cycle) to avoid additional latency.

In *Caeva*, we avoid sending the coordinator identity/notification cycle to *every* aggregator. We do this by relying on the fact that typically, event messages have topological locality (e.g., a fire is seen by publishers in the same region of the network). First, *Caeva* informs only the active aggregators of the coordinator identity/notification cycle. The passive aggregators then receive this information lazily, from its associated active aggregator, if and when they receive a message pertaining to the event. This means that most passive aggregators are oblivious to most events.

Dynamic and Adaptive Aggregator Placement

We now describe our adaptive passive aggregator placement algorithm. This algorithm adapts the placement of the passive aggregators based upon the patterns of published event messages. This algorithm executes continuously in the background, and at the conclusion of each event, it decides whether to alter the positions of the passive aggregators or to maintain the current placement. When altering the PA placement, the PAs are moved by only one hop at each step. In other words, at the end of an event, the algorithm decides one of three things: (1) maintain the current PA placement; (2) move the PAs one hop away from the active aggregators (towards the edge of the routing AG); or (3) move the PAs one hop towards the center of the routing AG. The decision is based on the estimated costs and benefits of each option.

Three types of brokers are involved in executing the algorithm, namely, the current set of PAs, the immediate upstream brokers of the current PAs (parents of current PAs) and the active aggre-

gator of the event under consideration. Each parent broker estimates the benefits and costs of moving the PA functionality from its children to itself (i.e., moving its downstream PAs one hop closer to the center), while each PA estimates the costs and benefits of moving the PA functionality to its children brokers (i.e., moving PAs one hop away from the center). The estimates from all PAs and parent brokers are consolidated at the active aggregator, which computes the cumulative costs and benefits of the three options and adapts the PA placement accordingly.

Now we discuss the formulations for estimating the costs and benefits for moving PAs one hop closer and one hop away from the center of the routing AG. First, we explain the cost and benefit formulae for moving PAs one hop closer to the center. Each parent broker uses these formulae to calculate the costs and benefits of moving PA functionality from its children to itself. Consider one such parent node pt_x. Let $CH(pt_x) = \{pv_1, pv_2, \ldots pv_Y\}$ be its children brokers (note that these nodes are a subset of the current $PvSet$). Let H denote the distance between the active aggregator and the current $PvSet$. For any broker b_i of the overlay, let $Pm(b_i)$ denote the number of messages of an individual event e_q published directly at b_i (i.e., published by publishers directly connected to b_i), $Fm(b_i)$ denote the number of messages of the same event forwarded by its downstream neighbors, and $Rm(b_i)$ represent the sum of $Pm(b_i)$ and $Fm(b_i)$. Let Nc denote the number of notification cycles for which the event e_q lasts ($Nc = \frac{dn(e_q)}{t_m}$, where $dn(e_q)$ denotes the total duration for which the messages pertaining to e_q are published and t_m denotes the length of the notification cycle.

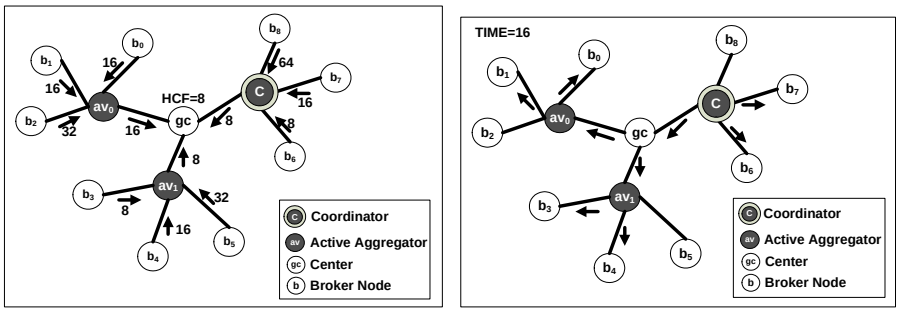
We now formulate the benefits of moving the PA functionality from $\{pv_1, pv_2, \ldots pv_Y\}$ to pt_x. If pt_x were to assume the PA functionality, it would send one aggregated message to the coordinator at the end of each notification cycle instead of $pv_1, pv_2, \ldots pv_Y$ individually sending an aggregated message at the end of each notification cycle. Furthermore, the aggregated message from pt_x would need to travel one hop fewer than the messages from the aggregated messages from the current PAs. Thus, the number of message hops saved over the entire duration is $Nc \times (H \times Y - (H - 1))$. Also, if pt_x assumes the PA functionality, the messages published directly at pt_x would be aggregated/eliminated immediately, thereby avoiding the need for these

messages to individually travel until the coordinator. Therefore the benefits of moving the PA functionality to pt_x is $BN(pt_x) = Nc \times (H \times Y - (H-1)) + Pm(pt_x) \times (H-1)$. However, there are also costs associated with moving the PA functionality to pt_x. Notice that if pt_x becomes the PA, all the messages received at $pv_1, pv_2, \ldots pv_Y$ have to travel one extra hop before being aggregated. Therefore, the extra overheads involved in moving PA functionality to pt_x is $CN(pt_x) = \sum_{pv_y \in CH(pt_x)} Rm(pv_y)$. Thus, the relative savings obtained by moving the PA functionality to pt_x is $SN(pt_x) = BN(pt_x) - CN(pt_x)$.

Through a similar reasoning, we can compute the costs ($CF(pv_i)$) and benefits ($BF(pv_i)$) of moving the PA functionality from an arbitrary passive aggregator pv_i to its Z child brokers $\{cp_1, cp_2, \ldots, cp_Z\}$, respectively, as $CF(pv_i) = NC \times ((H+1) \times Z - H) + Pm(pv_i) \times H$ and $BF(pv_i) = Fm(pv_i)$. Thus, the savings obtained by transferring PA functionality to child brokers of pv_i is $SF(pv_i) = BF(pv_i) - CF(pv_i)$. Note that SN and SF can acquire negative values.

At the end of culmination of an event, the coordinator obtains the SF values from each current passive aggregator and SN values from each parent broker of current passive aggregators. It then sums up the various SN values to obtain the cumulative SN (CSN) value, and it computes the cumulative SF (CSF) value as the sum of various SF values. These values are used in adjusting the PA placement as follows. If $CSF \geq \delta$ then PAs are moved one hop away from the center. If on the other hand, $CSN \geq \delta$ then PAs are moved one hop closer to the center. If neither condition holds, then PAs are maintained at their current positions.

One issue that still need to be addressed is that of preventing thrashing (PAs continuously alternating between two positions). We achieve this by introducing an extra condition. The PA adaptation direction can be reversed only when the estimated savings are higher than the savings in the previous adaptation that brought PAs to their current position. Concretely, suppose in the last adaptation the PAs moved one hop closer to the center and the estimated cumulative savings (CSN) was μ. The PAs move back to their earlier positions (one hop away from the center) only if the estimated savings (CSF) of the current adaptation is higher than μ. Otherwise the PAs are



(a) Upward Preference Propagation (b) Selective Notification Dissemination

Figure 4.2: Illustration of Customized Notification Scheme

maintained at their current positions even though $CSF \geq \delta$. An analogous strategy is adopted for moving the PAs closer to the center when they had moved away in the last adaptation.

4.2.3 Subscriber-Customized Notification Cycle

Finally, we describe how *Caeva* allows each subscriber to choose its notification cycle duration. In the *Caeva* prototype, a subscriber can choose its notification cycle duration in integer multiples of minimum notification duration (md). As mentioned before, a client specifies this at subscription time. A simple and naive way of implementing a customized notification cycle would be to hoard the notification messages sent out by the coordinator at the broker that is directly connected to an arbitrary subscriber s_i. The broker would send out notification messages to s_i at appropriate instances of time. However, this leads to unnecessary messaging within the overlay.

Instead, *Caeva* sends a notification through a path of the routing AG only if there is a subscriber downstream that should receive the notification at current instance. This is achieved by a combination of *upward propagation of subscriber preferences* and *selective downward dissemination of notifications*.

Upward Preference Propagation: The subscriber chooses its notification cycle duration in integer multiples of md. An arbitrary leaf broker of a routing AG, say b_k, may have multiple subscribers with different notification cycle durations. The edge broker calculates the highest common factor (HCF) of the notification cycle durations of the subscribers directly attached to it. This value indicates the period at which b_k should receive notification from its upstream node. Broker b_k sends this value to its upstream neighbor. A non-leaf broker, say b_j, calculates the HCF of the values sent by its downstream neighbors and the notification cycle durations of the subscribers directly attached to it, and propagates to its upstream neighbor. This is the period at which b_j should receive notification from its upstream neighbor. This process culminates at the graph center, which performs the same computation. The result is the HCF of the notification cycle durations of *all* subscribers being served by the routing AG. This value is maintained at the center and is used by the coordinator as the cycle duration for issuing aggregated messages. Figure 4.2(a) illustrates the upward preference propagation mechanism on a routing AG with 13 brokers. The HCF of the notification durations of all subscribers is 8, which is used as the cycle duration for issuing aggregated messages.

Selective Notification Dissemination: As described in Section 4.2.2, at the end of each cycle the coordinator obtains partially aggregated messages from various aggregators and merges them to create a notification message. However, the aggregated message at the end of a particular cycle needs to be sent only if subscribers depend upon their notification cycle preferences. Thus, instead of blindly sending the aggregated message through the routing AG, the coordinator checks which of its neighbors should receive notification at the current time and sends the aggregated message only to them. The intermediate brokers and the leaf brokers also work in a similar fashion. When a broker b_j receives an aggregated message from its upstream neighbor, it sends the message to only those downstream neighbors (if any) and subscribers (if any) that are due to receive the message at the current time. If the message is not sent to at least one downstream neighbor or subscriber, b_j maintains the message in a temporary buffer. While sending a message to a downstream broker, say b_k, b_j sends all those fields that have not been sent to b_k but are available currently at b_j. The

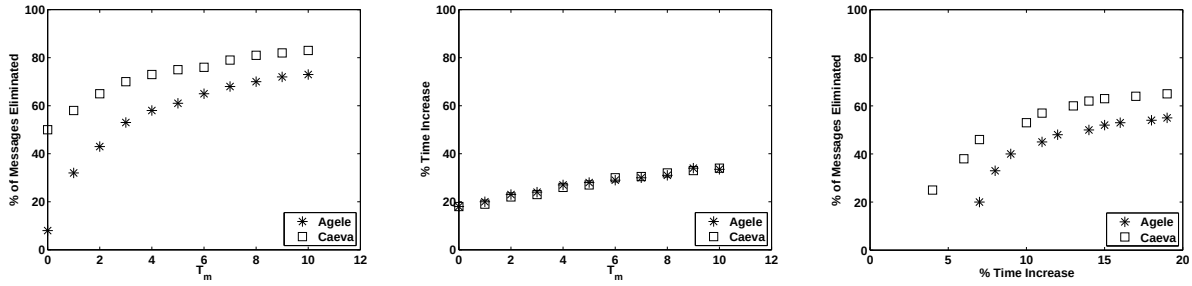


Figure 4.3: When T_m varies, percentage of messages in broker overlay suppressed (left); time increase (center). On the right, tradeoff between delay and percentage of messages eliminated.

exact same process is followed when sending messages to subscribers. Figure 4.2(b) demonstrates the selective notification dissemination technique at time 16. Notice that av_2 sends the aggregated message to b_4 and b_5, but not to b_6.

4.3 Experimental Results

Caeva has been implemented on top of the *Siena* pub-sub infrastructure [3]. We have performed several experiments to study the performance of *Caeva*. The goals of our study are two fold: **(1)** Evaluating the effects of *Caeva* on the broker overlay; and **(2)** Evaluating the effects on resource-constrained subscribers;

4.3.1 Setup

Our experiments were set up as follows. In all cases we use a random graph topology. Each complete event in our experiments consists of 20 fields, including the event key. In published messages, the number of fields that holds valid data varies from 1 to 10. The number of messages pertaining to an individual event can vary, and they are generated in the following manner. Each publisher of a particular event generates messages pertaining to that event according to a Poisson process. The event duration is chosen to be a maximum of 100 time units. In our experiments, all nodes

subscribe once and for any event. The particular event and associated field names are selected according to a uniform random distribution.

In our experiments, we use a merge threshold (denoted T_m) and a redundancy threshold (denoted T_r, and this value is fixed in our experiments). T_m is the notification cycle (defined in the Section 4.2.3), T_r is the amount of time messages are buffered at broker nodes in an attempt to discard later redundant messages.

Overall, an experiment is defined by its spatial locality for publishers, redundancy ratio for messages, and values for T_m and T_r. Spatial locality can be defined using the median distance between all pairs of publishers. However, in practice, it is difficult to set these distances in *Caeva* (due to limitations in *Siena*). Therefore, we vary the spatial locality between three configurations: (1) completely local, where all publishers reside at the same point in the graph; (2) partially local, where there are a few clusters of publishers, and (3) non-local, where all publishers are at different points in the graph.

In addition, the messages sent by the publishers for a given event can vary in their redundancy. We define the *redundancy ratio* for an event as F_r/M_t, where F_r denotes the number of messages whose fields are a subset of the fields previously sent, and M_t is the total number of messages sent. In our experiments, both T_m and T_r ranged between 0 and 10 simulated time units, such that $T_m \leq T_r$.

In the experiments below, we generally measure three different implementations. *Siena* provides the baseline. *Agele* is our previous system [44], on which *Caeva* is based; *Agele* is centralized, static, and uses one center node for aggregation, while *Caeva* is distributed and adaptive. Generally, we examine three important metrics: (1) percentage of the messages that are suppressed (by merging or duplicate elimination), (2) extra time that is added due to buffering at aggregators (measured by when the complete event is received), and (3) complete events and amount of data that subscribers receive.

4.3.2 Effect on Broker Overlay

We begin by investigating the effect that *Siena*, *Agele*, and *Caeva* have on the broker overlay. Here, we are interested in the total messages in the system. For this experiment, we use a random topology, low spatial locality, and the medium redundancy ratio. For *Agele* and *Caeva*, we vary T_m in the experiments. All results are relative to *Siena*.

Figure 4.3 shows the results. Because *Siena* does not handle redundant and partial event messages, it incurs more messages than either *Agele* or *Caeva*. In particular, *Caeva* eliminates up to 80% of the messages in the overlay. Comparing *Caeva* to *Agele* shows that the former suppresses more messages as T_m increases. This is because *Caeva* eliminates messages at the passive aggregators, which are closer to the publisher. This has two beneficial effects: (1) it takes additional message load off of broker nodes in between the passive aggregators and the coordinator, and (2) it can, in some situations, take additional message load off of brokers in between the coordinator and the subscribers. The latter point is somewhat subtle: if a message is not eliminated at the passive aggregator, then it proceeds to the coordinator. The coordinator may eliminate it, but it is possible that T_m is sufficiently small that it is *not* eliminated.

The center graph in the figure shows a time increase (for completed events) for both *Caeva* and *Agele*. Additionally, as expected, the relative time increase is larger with larger T_m. One item to note is that *Caeva* and *Agele* have essentially the same overhead. This is by design—the passive aggregators flush their buffered messages such that they reach the coordinator just in time to be flushed to the subscriber. (The small difference is because the coordinator in *Caeva* is a different broker node than the center in *Agele*.) The right graph shows similar information to the left and center graphs, but specifically shows the tradeoff between increased latency and the number of messages eliminated.

Next, we study the effect on the broker overlay when the spatial locality of the publishers as well as the redundancy ratio vary. We used the spatial localities and redundancy ratios specified above. In the graph, the first letter refers to the spatial locality; “H” for completely local, “M” for partially local, and “L” for non-local. The second letter refers to the redundancy ratio; “H” for a

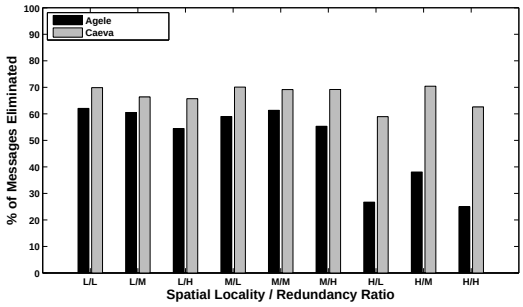


Figure 4.4: Percentage of messages in broker overlay suppressed when spatial locality and redundancy ratio vary; the first letter indicates the locality, and the second the redundancy ratio

redundancy ratio of 85%, “M” for 50%, and “L” for 20%. In these tests, T_m and T_r are both 10. Figure 4.4 shows the results. We see that as the spatial locality of the publishers increases, the advantage of *Caeva* increases over *Agele*, in terms of message load in the broker overlay. This is because more of the published messages are directed to the same passive aggregator, which eliminates some of them.

We note that many scenarios of publisher locality and redundancy ratio are possible. For example, a news bulletin occurring at night would potentially lead to widely distributed publishers, whereas an accident during rush hour would likely lead to mostly localized publishers. *Caeva* is actually the best choice for all of these cases, though its advantage increases with more locality in space and time. The one disadvantage of *Caeva* relative to *Agele* is that it is more complex and involves more broker-broker communication.

4.3.3 Adaptive PA Placement

Table 4.1 shows the number of messages for different numbers of publishers for both the static and adaptive algorithms. For the static algorithm, the passive aggregators can reside at several different places; we show both the minimum and the maximum. This experiment uses publishers with similar characteristics. The key point is that the adaptive algorithm is always close to as good

Publishers	**Static**		**Adaptive**
	Min	Max	
3	101,796	125,714	102,583
7	147,913	220,126	150,239
31	181,189	232,420	197,141
255	203,241	227,747	211,375

Table 4.1: Number of messages for different numbers of publishers for both static and adaptive algorithm

Varying Publishers	**Static**		**Adaptive**
	Min	Max	
Uniform	153,847	203,474	153,385
Nonuniform	293,265	361,287	266,722

Table 4.2: Number of messages for different numbers of publishers for both static and adaptive algorithm when publishers have nonuniform characteristics

as the minimum and avoids the large penalty of choosing the maximum. Keep in mind that the static algorithm requires a single placement, and without application-specific knowledge, it is possible that a bad placement might be chosen.

Next, Table 4.2 shows the same attributes, but compares the uniform and nonuniform publisher case. It is clear that for nonuniform publishers, the adaptive algorithm is significantly (10%) better. This is because when publisher characteristics change, the static algorithm cannot change. On the other hand, the adaptive algorithm changes based on these characteristics.

4.3.4 Effect on Subscribers

We now look at the effect of *Siena*, *Agele*, and *Caeva* on subscribers. The metric that we study is number of completed events. Here, we assume, reasonably, that subscribers are mobile devices that have scarce computing resources. We use a simple model where each subscriber has a buffer to

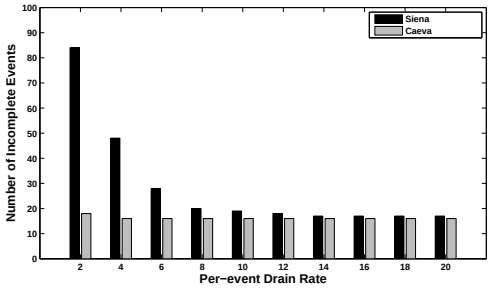


Figure 4.5: Incomplete events when varying the per-event drain rate; the per-event buffer size is fixed at 20 fields

store incoming messages. The messages will be processed and consumed by the application (or the user) at a certain rate. With the intention of avoiding interference among multiple events that could affect consistency of results, we allocate and maintain individual buffers for each active event at a particular subscriber. The parameter *per-event buffer size (sz)* controls the size of the individual event buffers. Similarly, *per-event drain rate (dr)* controls the rate at which the messages of an individual event are consumed (removed from the respective buffer) by the application or the user on the subscriber device. Any messages that would cause buffer overflow are dropped. Note that if there are f active events at a subscriber the total allocated buffer size is $f \times sz$ and the cumulative message consumption rate is $f \times dr$. In our experiments, we study the effects of sz and dr on the number of completed events in *Siena* and *Caeva* (*Agele* is similar to *Caeva*). For this experiment, we use non-local publishers and a medium message redundancy ratio.

Figure 4.5 shows that with small per-event buffer sizes and small per-event drain rates, there is a significant difference between *Caeva* and *Siena*. This difference, as expected, decreases as the buffer size and drain rate increase. Keep in mind that while the buffer size we used is small, each subscriber only subscribes to one event, and the number of fields in an event is small. In a real system, all of these things would be much larger, but the fundamental issue remains: there will be situations where a buffer cannot hold all messages arriving at a subscriber.

CHAPTER 5

COMPLEX EVENT DETECTION ON DELAY TOLERANT NETWORKS [1]

[1] Jianxia Chen, Lakshmish Ramaswamy, David K. Lowenthal and Shivkumar Kalyanaraman. "Comet: Decentralized Complex Event Detection in Delay Tolerant Networks". Submitted to the 28th IEEE International Conference on Data Engineering (ICDE 2012), 07/22/2011

In this chapter, we discuss a novel, multi-level framework, called *Comet*, for efficient and scalable complex event detection (CED) in delay tolerant networks (DTNs).

5.1 ABSTRACT

Complex event detection is fundamental to monitoring applications. Current complex event detection (CED) techniques are targeted for continuously connected, high-bandwidth, Internet-based environments, and are mostly centralized. However, event monitoring applications are becoming increasingly important in domains such as deep-space, warfare and rural, where lack of infrastructure to support the Internet has led to the development of the delay tolerant networking (DTN) paradigm. DTNs are characterized by decentralization, long delays and frequent disruptions, which necessitates a complete, end-to-end re-design of CED techniques. In this chapter, we create Comet, which provides efficient and scalable CED for DTNs. The novelty of Comet is that it addresses efficiently all pertinent CED issues in a decentralized environment. 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. Comet finds near-optimal individual CED plans through two novel heuristic planning techniques: multi-level push-pull conversion and virtual CED tree creation. Additionally, Comet eliminates redundancy that occurs when complex events contain common primitive events; the redundancy is eliminated by efficiently merging the respective CED trees. Performance results show that Comet reduces cost by up to 89to pushing all primitive events and over 60single-level exhaustive search algorithm.

5.2 COMET OVERVIEW

As mentioned earlier, a distinguishing feature of Comet is that it supports multi-level CED in which multiple nodes can participate by performing parts of CED. In other words, each node involved in the CED process detects a sub-complex event (sub-CE) of the original CE. Suppose a node V_f is involved in the detection process of ce_i, the sub-CE detected at V_f is represented as

sce_i^f. Comet consists of two major components, namely a *CED planner* that creates a cost effective detection plan based upon known or estimated statistics, and an *execution and adaptation engine* that executes a CED plans and adapts it to cope with various dynamics. In this dissertation, our focus is on the *CED planner*.

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? **(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? Finally, **(4)** if multiple CEs share common sets of PEs, when and how is redundant communication and computation avoided? In particular, which DTN nodes should host the common sub-CEs corresponding to the shared sets of PEs, and what push-pull strategies should be adopted for common sub-CEs? 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.

At a very high level, our planner is comprised of two modules, namely, a *individualized multi-level CED planning module* for producing cost-efficient, multi-level, push-pull plans for each individual CE, and a *redundancy avoidance module* for leveraging PEs shared among multiple CEs by incorporating common sub-CE detection processes.

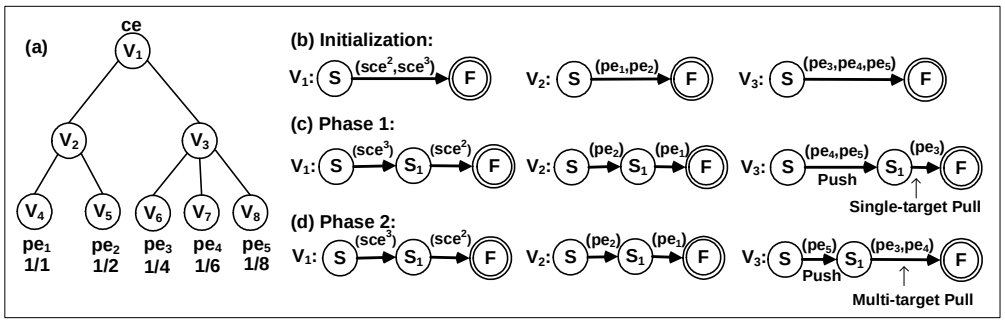


Figure 5.1: Illustration of Multi-level Push-Pull Conversion

The CED tree forms the edifice of both of these modules. 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*.

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 [55] 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.

5.3 Individualized CED Planning in Comet

As mentioned earlier, our basic multi-level planner produces efficient multi-level plans for detecting individual CEs. In essence, this module treats each CE independently (without considering any overlap with other CEs) and produces near-optimal plan for that CE. If the CEs that are being concurrently detected do not share any PEs, the set of plans obtained by applying this multi-level planning module to each of the CEs will be close to optimal with respect to cumulative detection costs over all the CEs in the system. This module itself has two novel components. Our first component addresses the challenges in extending the push-pull conversion-based planning strategy to multi-level CED trees. The second component creates multiple virtual trees for a given CED tree to overcome the potential suboptimality caused by operating at link granularity (see Section 5.3.2). Comet creates a set of virtual CED trees, executes the push-pull component on each topology, and selects the best plan among them.

5.3.1 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 5.1).

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 fre-

quencies. 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 cost estimation model [20] 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 [56].

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.

5.3.2 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 5.2-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

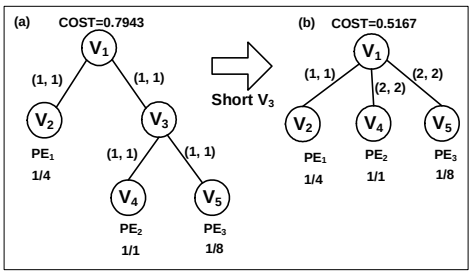


Figure 5.2: Virtual Topology Creation via Shorting

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 5.2-b indi-

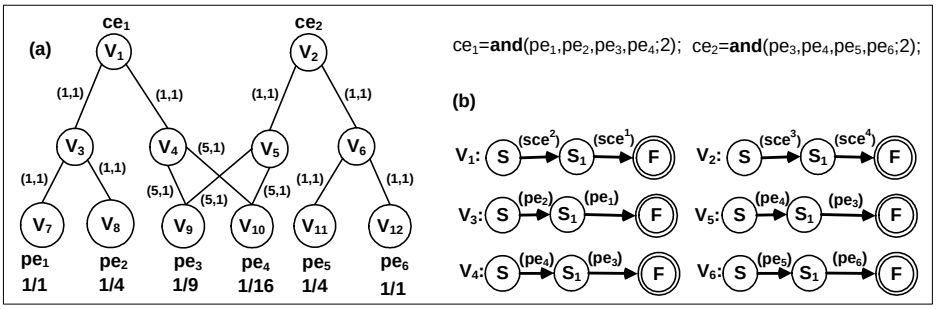


Figure 5.3: Illustrating the Need for Avoiding Redundancies in Multi-level CED

cates 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 5.2-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).

5.4 AVOIDING REDUNDANCY IN COMET

Comet's individualized CED planner evolves plans for each CED independently without considering other CEDs that may be concurrently detected. As our experiments in Section 5.5 demonstrate, this yields highly efficient plans for each individual CE. However, this *individualized* approach may lead to redundant computation and communication in scenarios where CEs share common constituent PEs. This may in turn lead to situations where individual CE plans are optimal, but the *cumulative detection costs* of the system may be suboptimal. The scenario in Figure 5.3 has two CEs, ce_1 and ce_2, with pe_3 and pe_4 shared between them. The detection plan for ce_1 and ce_2 obtained through our planner is shown in the figure. In fact, these are the optimal individual plans for these two CEs. The cumulative detection cost of the two CEs is 2944 over 2000 time units. On the other hand, detecting the sub-CE (**and**$(pe_3, pe_4; 2)$) either at node V_4 or V_5 and then sending the detected event instances to both V_1 and V_2 yields a cumulative cost of 1999 over 2000 time units. This illustrates the need for leveraging common parts of CEs to avoid redundant communication and computation.

Although Akdere et al [20] have proposed a technique to leverage common parts of CEs, their technique is designed for a centralized CED system with the assumption that all the CEs are entirely detected at the same destination. This assumption obviously does not hold for a multi-level, decentralized environment. Thus, we need a technique that works in conjunction with distributed, multi-level CED plans and can leverage common parts between CEs that may have different destinations. Broadly, our idea is to avoid duplication by instantiating common processes to detect the sub-CEs corresponding to each set of shared PEs (e.g., **and**$(pe_3, pe_4; 2)$ in Figure 5.3). Each such common sub-CED process will be hosted on a DTN node and the detected sub-CE instances from the process will be sent to destinations of CEs sharing the sub-CE (or intermediate nodes performing next stage of detection). In effect, we merge the CED trees to obtain a directed acyclic graph (DAG) where the nodes hosting the common sub-CED processes will have multiple parents.

Designing a concrete technique to realize this idea poses several challenges. First, given a set of CE definitions (specifying the destination and constituent PEs for each CE), we need to decide

whether it is even beneficial to avoid duplication. In not all circumstances is it worthwhile to do so. For instance, if the destinations of two CEs sharing a set of PEs are far apart with respect to the underlying DTN topology, it is better to retain individual CED plans despite incurring overheads of duplicate communication. Second, we need to decide which node to employ for detecting the common sub-CE. The chosen node should minimize the cumulative detection costs. Finally, the original detection plans of CEs may embed conflicting sub-plans for detecting the common sub-CE. In Figure 5.3, for example, the original plan for ce_1 specifies pe_4 to be pushed and pe_3 to be pulled, whereas the original plan for ce_2 requires both pe_3 and pe_4 to be pushed due to stringent delay tolerance. We need to reconcile such conflicting plans for the common sub-CE. In many cases, reconciliation will necessitate modification to the non-common parts of the CED trees as well.

We propose a two step technique to address these challenges. For the first step, we propose a novel heuristic-based algorithm to select the best node (node that minimizes cumulative detection costs) for hosting the detection process corresponding to the common sub-CE (sub-CE comprising of the shared PEs). In doing so, we also decide whether instantiating a common sub-CED process is beneficial from a cumulative detection cost standpoint. The original CED trees are merged to utilize the results from the node hosting common sub-CED process (this node is henceforth referred to as *common junction node* or *CJN* for short). In the second step, we use a conservative approach to determine the detection plan for the common sub-CE (to avoid any delay tolerance violations) and then apply the previously-described push-pull conversion algorithm to the non-common parts of the reconstructed CED trees. We now briefly discuss each step.

5.4.1 CJN Selection and CED Tree Reconstruction

We now explain our heuristic-based CJN selection algorithm. Once the CJN is decided, we determine whether it is at all beneficial to merge CED trees so as to leverage common sub-CE detection at the CJN. We also outline the CED tree reconstruction mechanism to utilize the CJN. Our primary objective in choosing a DTN node as CJN is to minimize the cumulative detection costs. Such a

hosting node could be a junction node in the original CED trees, or a node outside of all CED trees (but nevertheless part of the underlying DTN). Towards this end, our algorithm takes into account both the cost factor and the delay of the paths from the sources to the destinations of the CEs via the node being considered as a candidate for CJN. The reason for considering cost factors is straightforward as they directly impact the cumulative detection costs. It is also important to consider the delays of the paths because we have to ensure that the merged CED trees along with new detection plans (discussed in the next subsection) honor the delay restrictions for all of the CEDs involved. By considering path delays, we increase the likelihood of producing a near-optimal plan in step 2.

Our algorithm starts by constructing a list of candidate CJNs. A DTN node must satisfy two conditions to be considered a candidate CJN: (1) It has paths to all the shared PE source nodes, and; (2) It has paths to the destinations of all CEs. Such a list is constructed through simple reachability analysis. Once such a list is constructed, we have to choose the node that minimizes the cumulative detection costs. The problem is that we cannot evaluate the cumulative detection costs for the candidate CJN node until the exact push-pull plan associated with the node is known. Determining the exact plan for each candidate CJN (by repeatedly executing step 2) is prohibitively expensive.

We address this problem by *estimating* the cost-delay benefits of choosing a candidate node as the CJN, irrespective of any specific detection plan. In fact, we quantify the cost and delay characteristics of the paths from the sources of the shared PEs to the CE destinations via the candidate node being considered. We utilize the h-factor (see Section 5.3) for this purpose. The eb value of a candidate node is the weighted sum of h-factors of the links from each shared PE source to the candidate node being considered and the h-factors of the links from candidate node to the destinations of the various CEs. The h-factor of each link is weighted by the frequency of the event that it is supposed to convey.

$$eb = \sum_{i} (f_i * Sh_i) + \sum_{j} (Dh_j * \sum_{i} f_i) \tag{5.1}$$

where Sh_i is the sum of the h factors of all the links along the path from the candidate CJN to the source of the shared primitive event pe_i, f_i is the expected number of pe_i instances per unit

time, Dh_j is the sum of the h factors of all the links along the path from the candidate node under consideration to the destination of complex event ce_j, and $\sum_i f_i$ is the expected number of occurrences for all shared PEs.

Our algorithm evaluates the eb values for all candidate CJNs. The node with the minimum eb value is to be selected as the CJN. However, at this stage it is still not decided whether to leverage shared parts of the CED trees. We do this by calculating the eb values of the junction nodes in the original (individual) CED trees corresponding to the shared PEs. If the sum of the eb values of the junction nodes of original CED trees is less than the eb value of the CJN, it means that leveraging shared events will not lower the cumulative detection costs, and the independent CED plans with original CED trees are retained. On the other hand, if the eb value of the CJN is lower, Comet leverages the common parts by reconstructing the CED trees. This is done by redirecting the shared PEs to the CJN, which detects the sub-CE corresponding to the shared PEs. The results of the sub-CE detection are then sent to the next stage of the detection process in each of the original CED trees.

Figure 5.4 illustrates our algorithm on the two CED trees from Figure 5.3. We assume that there are direct links from V_4, V_5 and V_{13} to both V_1 and V_2 making them candidate CJNs (for better clarity, the figure does not show all the links from the underlying DTN). The eb values of V_4, V_5 and V_{13}, corresponding to the merged CED trees are indicated next to them. The nodes V_4 and V_5 were part of the two original CED trees detecting the sub-CE **and**$(pe_3, pe_4; 2)$. The eb values of these nodes corresponding to the independent CED trees are indicated in the parentheses. V_{13}'s eb value is the lowest. Furthermore, it is also less than the sum of eb values of V_4 and V_5 from the two independent CED trees. Therefore, V_{13} is selected as the CJN, and the two CED trees are merged. The resulting DAG is shown with bolder lines.

5.4.2 Push-Pull Conversion on Reconstructed CED Trees

Once the CED trees are reconstructed, the next step is to determine the actual push-pull plans for the DAG obtained after merging the individual CED trees. We do this by first considering the part

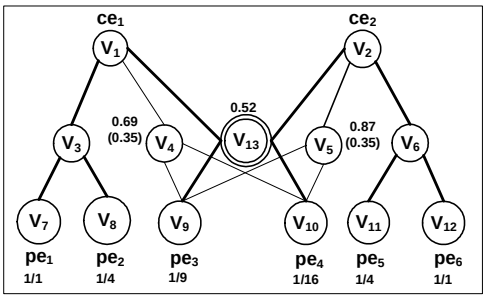


Figure 5.4: CJN Selection and CED Tree Reconstruction

of the DAG corresponding to each CE (including CJNs that are involved in detecting common sub-CEs). We run our original push-pull conversion algorithm independently for each CE. At the end of this step we have a plan for all CEs, within each of which is embedded a plan for the sub-CEs that it shares with other CEs. However, notice that if a sub-CE is common to k CEs, it may have up to k distinct plans. The question is which of these plan should be chosen. We use a conservative approach and choose the plan with lowest delay for the common sub-CE. The rationale for this choice is that we have to ensure that delay restrictions for all CEs containing the sub-CE are honored. Also, if the two CEs have different time windows in their CE definition, the selected sub-plan should use the larger time window for the pull requests. This ensures that we do not miss any CE instance. Our choice of the lowest delay plan may create additional delay slack in some CEs whose plans had embedded a higher delay plan for the sub-CE. This slack may allow us to further reduce the cost of such CEs. This is done by executing the push-pull conversion algorithm on the non-shared part of such CED trees, with the shared sub-CE parts of each tree being fixed [2].

5.5 Experimental Evaluation

We have implemented both *Comet* and our DTN simulator in Java. The DTN simulator simulates the DTN model described in Section 2.2. The simulator contains a number of DTN nodes, each

[2] An incremental approach can used for re-planning to reduce the computational effort involved

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 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 2.2). 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).

5.5.1 RESULTS

In the first set of experiments, we exclusively evaluate Comet's individualized multi-level CED planning mechanism. Therefore, for these set of experiments, we assume that the CEs do not have any common PEs. We compare Comet's individualized multi-level CED planning mechanism to three other algorithms. The baseline plan is *All-Push*, where all events are pushed to the CE destination 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. [20]. Note that it is optimal only for centralized solutions in which the CED plan is only executed at the destination, so Comet, with its multi-level nature, can outperform this notion of *Centralized Optimal*. The *Centralized Heuristic* plan is our adaptation of the single-level heuristic algorithm suggested by Adkere et al.[20]. In implementing *All-Push, Centralized Optimal* and *Centralized*

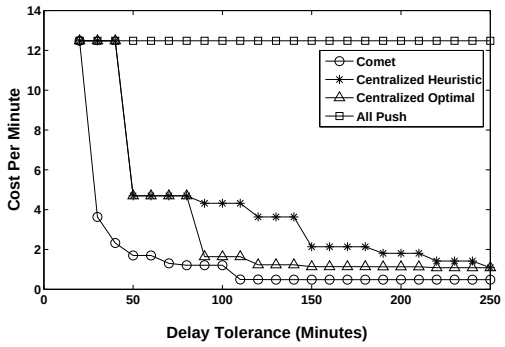


Figure 5.5: Performance with nonuniform cost and uniform delay per link.

Heuristic, we use the most cost-and-delay efficient paths (guided by h-value) for transferring PE instances to their respective destinations.

Figure 5.5 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. The other two algorithms—*Centralized Optimal* and *Centralized Heuristic*—are able to filter out some PEs. Comet is superior to both because its multi-level nature allows PEs to be pulled from the intermediate nodes closer to their respective sources, which filters out extraneous PE instances. Again, *Centralized Optimal* and *Centralized Heuristic* only pull events at the destination. Note that creating a multi-level optimal algorithm is infeasible because it is exponential in the number of links.

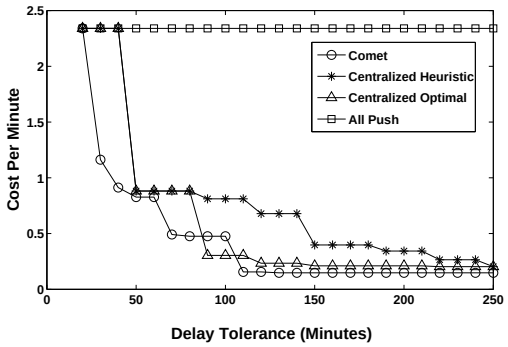


Figure 5.6: Performance with uniform cost and uniform delay per link.

Figure 5.6 shows results for a similar experiment as was shown in Figure 5.5, except that the bandwidth 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 from nodes closer to the source is lower. For this narrow range of delay restrictions, there are some centralized plans that outperform multi-level plans. As *Centralized Optimal* exhaustively explores the solution space of the centralized plans, 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 delays for different subtrees, which will eventually fully utilize the potential concurrency in detecting CEs.

Figure 5.7 shows results for the same experiment as discussed above, except that the link delay is now nonuniform. 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 5.5.

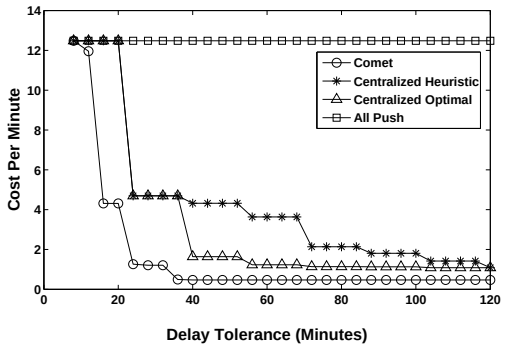


Figure 5.7: Performance with nonuniform delay per link for nonuniform cost per link.

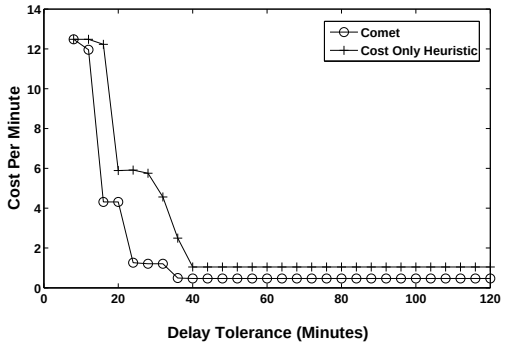


Figure 5.8: Performance of Comet with different heuristics. The cost per link is nonuniform.

Next, Figure 5.8 shows results for the same two experiments as discussed in Figure 5.7, except that the ordering of status changes in our multi-level push-pull conversion technique are determined using only cost rather than the ratio of cost to delay. 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 delay for ordering potential status changes in Comet. On one hand, using purely cost, irrespective of the change in delay, may cause Comet to choose pull operations that can cause significant delay increases and also leads to fewer pull operations elsewhere in the plan due to the delay tolerance. On the other

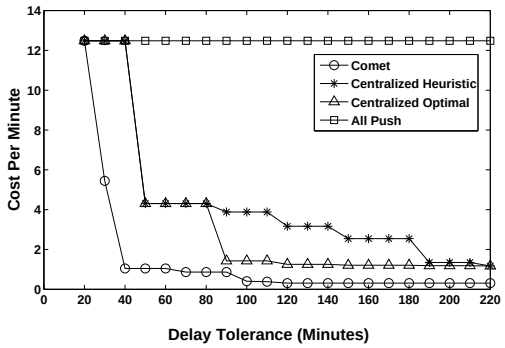


Figure 5.9: Performance where the degree of the junction nodes is varied (randomly) from 1 to 3.

hand, using the ratio of cost reduction to delay better balances the change of both cost and delay. It also can leverage the potential of pull concurrency, which can in turn lead to cost reduction with only a small delay penalty.

Figure 5.9 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.

Figure 5.10 shows the impact of the number of levels in the CED tree on the cost of the detection plan. When the number of levels of the CED tree increases, the cost decreases. This is due to the multi-level sub-CED 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 delay due to the long turnaround time of 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 delay between the junction node and the source. Note that at the same time, the cost of centralized optimal remains

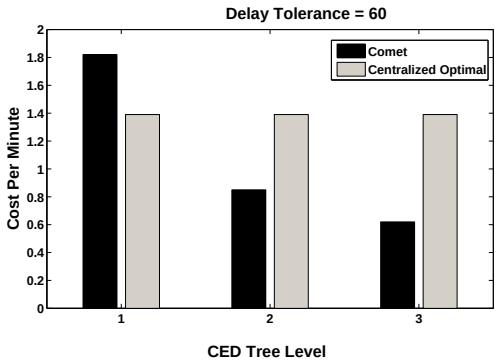


Figure 5.10: Performance of Comet with different CED tree levels.

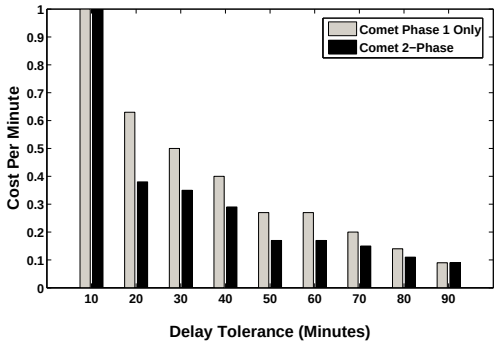


Figure 5.11: Benefit of using two-phase algorithm in Comet.

constant, because centralized detection plans do not utilize the junction node to further reduce the plan cost and delay.

Figure 5.11 shows the benefit of the two-phase push-pull conversion algorithm of Comet. The concurrent pull phase of Comet (the second phase) further explores the concurrency of pull operations, especially when the delay tolerance is modest. Most of the time, the two-phase algorithm results in a significant cost reduction compared with the algorithm with only conversion of pushes to single target pulls (the first phase). The second phase further reduces the cost by converting more pushes to pulls, but without a significant delay 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 available push operations.

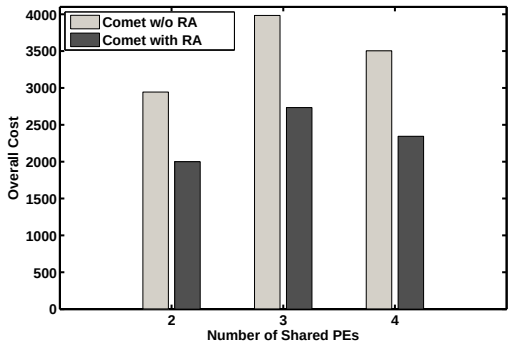


Figure 5.12: Performance on Different Number of Shared PEs

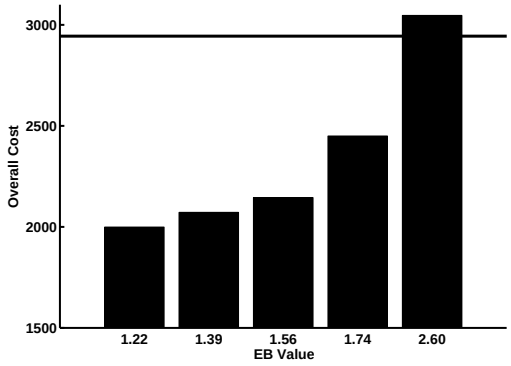


Figure 5.13: Comparison of Different Hosting Nodes (2 Shared PEs)

In the second set of experiments, we evaluate Comet's redundancy avoidance (RA) mechanism. First, in Figure 5.12, we compare two versions of Comet—one in which the redundancy elimination module (represented as Comet with RA) is enabled and the other in which the RA module is disabled (represented as Comet without RA)—with different numbers of common PEs among two CEs. Results show that Comet's redundancy avoidance algorithm can successfully reduce the overall detection cost when the number of common PEs varies.

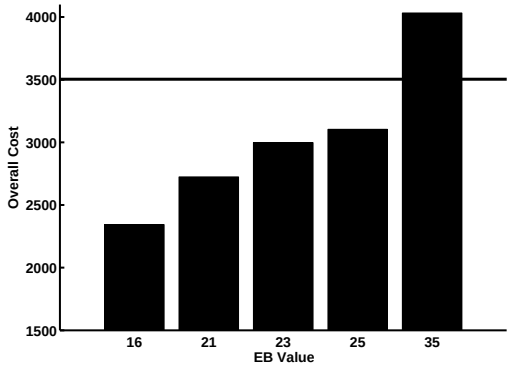


Figure 5.14: Comparison of Different Hosting Nodes (4 Shared PEs)

Figure 5.13 evaluates the eb value model for selecting the candidate hosting nodes. The horizontal line in the figure indicates the cumulative cost for Comet without RA. The corresponding eb value for this case is 2.08. The overall detection costs for different candidate hosting nodes with different eb values are presented in the figure. For the candidate hosting nodes with eb value lower than 2.08 (Comet without RA case), the overall cost is significantly lower than the Comet without RA case. Note that, for a candidate hosting node with eb value greater than the eb value of Comet without RA case, its overall cost would be higher than the Comet without RA case case. Generally, the lower the eb value, the lower the resulting overall cost. The Comet heuristic algorithm will choose the candidate hosting node with the lowest eb value (1.22), which results in the lowest cost (1999). Figure 5.14 shows similar experiments with 4 common PEs; the results are similar.

CHAPTER 6

LITERATURE REVIEW

Our work in this dissertation is directly related to publish-subscribe, distributed stream processing, complex event processing and delay tolerant networks. In this chapter, we survey the related work of event monitoring services in these areas.

6.1 PUBLISH-SUBSCRIBE

Publish-subscribe (pub-sub) systems have continued to be an important research area over several years [6, 2, 57, 3, 58, 9, 59, 60, 61, 62, 8, 5, 4, 7, 63]. On this basis of how subscriptions are specified, pub-sub systems are classified into topic-based [6, 64], content-based [2, 3, 5, 65], type-based [27], and type- and attribute-based [7] categories. From an architectural standpoint, distributed pub-sub systems [1, 2, 3, 66, 60, 8, 4] provide significantly better scalability than their centralized counterparts [67]. As mentioned in chapter 3, the subscription mechanism of *Agele* is similar to type and attribute-based subscription model [7], and it adopts a distributed broker framework.

Over the past decade, various aspects of pub-sub systems have been widely studied including subscription mechanisms, architectures, quality-of-service, mobility, and reliability [6, 68, 2, 3, 69, 58, 70, 9, 71, 72, 7, 63]. Surprisingly, the issue of redundant and partial event messages, which are very common in settings with human participants, has received little research attention. A few researchers have considered the problem of *exact* duplicate elimination [63, 73, 74]. However, most solutions are simplistic with performing duplicate elimination at the subscribers being the most common approach [63]. The XTreeNet system [73] uses an in-network duplicate elimination scheme. The scheme has two major limitations. First, it requires each node in the tree to maintain a

cache of messages that has recently passed through it. Because nodes often participate in multiple trees, they need to store large number of messages for this scheme to be effective. Second, the technique is not effective in reducing message traffic due to duplicates originating from different regions of the overlay. Thus, the system is not able to provide any guarantees to the subscribers or offer them flexibility with respect to the degree of duplicate elimination or the notification times. Our definition of redundancy is broader in the sense that an event message is considered to be duplicate if the information it carries has already been obtained by aggregation of other messages, even though it was not an exact match to any of the previous messages. Thus, our duplication elimination is more powerful. By designating specific event gatherers for every routing AG, we eliminate the need for message caching at each node in the overlay. Furthermore, our technique is effective even when the messages originate in different regions of the network. To the best our knowledge, our work with *Agele* was the first system to consider incomplete (partial) event messages. *Agele* is a centralized system that uses a center node to aggregate all messages; i.e. there is one, fixed active aggregator and no passive aggregators. In addition, *Agele* is static; the notification cycle is fixed over the entire system. *Caeva* is much different; it is distributed and therefore scalable, it allows flexible, adaptive placement of passive aggregators as well as a flexible choice of the notification cycle for each subscriber.

In the context of multicast routing, Thaler and Ravishankar [51] propose heuristic-based strategy for finding the graph center, which works in multiple rounds. There are two main differences between their algorithm and ours. First, being a heuristic-based approach, their algorithm may not always locate the exact center. Second, and more importantly, although their scheme does not require a global view of the overlay topology, it assumes that in each round the center knows about all nodes in the multicast group. By contrast, our algorithm does not require centralized membership information, and it always discovers the exact center of the routing AG. The scheme by Song [50] requires each node to first discover the identity of all other nodes, and then execute the all pairs shortest path algorithm. Unfortunately, this straightforward distribution strategy

imposes significant computation and communication overheads at all nodes in the network, thus making it impractical for our application.

6.2 Distributed Stream Processing

Our work in this dissertation is directly related to the area of distributed stream processing [10, 11, 12, 13, 14, 15, 16, 17]. Compared to the traditional DBMS whose data is relatively static, the data in distributed stream processing systems is extremely mobile. In such systems, DBMS is active and human is passive, while in traditional DBMS, human is more active and DBMS is passive [10]. Existing work focuses on different aspects of distributed stream processing systems [13, 14, 16, 17]. The Aurora [10] and Borealis [11] study the techniques to process the data streams on single or multiple sites with the emphasis on load balancing, runtime query and result modification, and QoS based optimization. The STREAM system [12] is trying to provide similar DBMS functionality on the data streams with special concentration on the memory requirements. TelegraphCQ [15] specializes on shared, continuous query processing over query and data streams.

The area of distributed stream processing [75, 76, 77, 54, 78, 79, 80] has similarities to event aggregation in decentralized pub-sub systems. In both these cases, data originating from the nodes of an overlay needs to be processed and delivered to a set of recipient nodes. However, there are also crucial differences between the two. First, in stream processing systems, the source nodes of various data streams are generally known when the query plan is evolved. Whereas in a pub-sub system, any publisher that has issued an advertisement can generate a corresponding event. Second, the data streams last for relatively long durations of time, and so do the data processing operators defined on these streams. Third, many of the stream processing systems assume a global view of the overlay topology. These characteristics justify and permit the heavy-weight, optimization-based query planning, operator placement, and adjustment strategies used by stream processing applications. The pub-sub environment, especially in community-oriented applications, is much more ad-hoc — publishers generate event messages in a non-continuous manner and at arbitrary points in time. Furthermore, each event is active for short duration of time, in the sense that the

messages pertaining an event are published in a short time window. Thus, the heavy-weight operator placement strategies are not appropriate for *Agele* and *Caeva*. In contrast to distributed stream processing systems, *Caeva* does not require a priori knowledge of event message sources, and its protocols and techniques are lightweight and dynamic.

6.3 Complex Event Processing

Complex event detection (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 [46]. Current work on CED has focused on two main issues, namely, reducing the computational overheads at the server [18, 19] and reducing the communication costs [20]. Wu et al [19] study NFA based optimization techniques to achieve faster complex event processing at the server-end on high volume event streams with long time windows. Ding et al. [18] exploit event constraints to optimize complex event processing over large volumes of business transaction streams. The plan-based CED technique [20] reduces the communication overheads of CED by intelligently pushing and pulling (through single and multi-target pulls) PEs. Our work also focuses on reducing the communication overheads. However, there are several crucial differences between Comet and these existing systems. First, the existing systems are designed for traditional, continuously connected networks. Comet, on the other hand is designed for intermittently connected DTNs. Second, most of these 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.

Complex event detection [20] also bears similarities to event aggregation. However, most of the current approaches to complex event detection rely upon apriori planning which assumes that the event sources are known before hand. The difference between our work and existing stream processing system is that existing work tends to focus on the complex event language and stream processing performance at the data stream sink, while we focus on reducing data transmission cost within the stream processing network by executing effective detection plans which utilize the

properties of the event streams and network to eliminate the transmission of unnecessary data. Adaikkalavan and Chakravarthy [81] discuss modeling and specification of incomplete events. The SASE+ [21, 22] processes multiple event streams trying to find certain pattern of correlation. It has a more descriptive event language to improve the expressibility of complex events. At the same time, a NFA model is applied to improve the performance of complex event processing at the event sink.

There is a crucial difference between data stream processing systems and complex event detection systems. As Akdere et al. [20] have noted, data stream processing systems operate on a continuous query paradigm and are expected to produce results constantly. Hence they rely exclusively on push-based data transfer. Complex event detection systems on the other hand are geared towards detecting certain events of interest. Unless such events occur, the system essentially does not produce any output. Thus, CED systems can utilize both push and pull data transfer modes. Furthermore, most of the current distributed data stream processing systems are designed for traditional, continuously-connected networks. While our work is based on delay tolerant networks, where long latency and intermittent connection are common.

6.4 Delay Tolerant Networks

Delay Tolerant Networks (DTN) has been an active area of research for the past few years [42, 82, 83]. One of the typical types of DTN is the interplanetary network (IPN) or deep space network, which is used for the data transmission among different type space vehicles and the ground station on earth. For example, in Mars exploration [41], Bundle Protocol [84, 33] has been widely used for the transmission of data in IPN. Recent develop in IPN includes testing the Bundle Protocol from space using the United Kingdom Disaster Monitoring Constellation, a multi-satellite earth-imaging low-earth-orbit sensor network where captured image are first stored on board each satellite and later downloaded to a ground station [85, 86, 87].

The major focus of DTN research is on routing and message dissemination algorithms [42]. Since there is no end-to-end path in DTN, messages are propagated in a store-and-forward manner.

Historical data and replication techniques are usually used in DTN routing, such as [82, 83]. There are also some designated nodes called data mule or data ferry to perform the major routing task, for example in [88, 89]. Recently social-based approaches are also applied in the DTN routing, for example [90, 91].

Although DTN has been the active research area in the past decade, there is very limited number of work on running the applications on the DTN. Research on this important aspect of DTNs has been limited to web applications and distributed file systems [92, 93]. This dissertation is a step towards closing this gap in the sense that it studies how complex event monitoring applications have to adapt when the underlying network is a DTN. To the best of our knowledge, we are the first to apply the plan based multi-level complex event detection techniques on delay tolerated network (DTN).

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this chapter, we summarize the challenges and our contributions to the event monitoring services and discuss the future research directions for event aggregation and complex event detection on delay tolerant networks.

7.1 DISSERTATION CONCLUSIONS

In real world event monitoring systems, especially those ones with human publishers, are potentially faced with event data that is fraught with various kinds of noise. Dealing with this noise effectively is critical, yet challenging both from data management and distributed computing perspectives. This dissertation presents our approach towards designing an efficient distributed broker overlay-based event monitoring system that eliminates redundant event messages as well as aggregates information from multiple messages corresponding to the same event.

We first introduce the concept of *event gatherer* as a designated broker of a particular routing graph that is responsible for eliminating redundant messages and merging messages containing partial event information through that routing graph. We show that in order to achieve high efficiency and low overheads, the event gatherer should be located at the graph center of the corresponding routing graph. A novel, completely decentralized algorithm has been presented for discovering the center of an acyclic broker network. The above ideas are incorporated into our system, *Agele*. We have performed several experiments for studying the performance of the *Agele* system under various conditions. Results show that the proposed techniques are effective and efficient, thereby demonstrating the viability of our approach.

In the extened work of *Agele*, we introduce the distributed event aggregation and redundancy elimination system: Caeva. Caeva uses a collaborative broker overlay to eliminate redundant messages and merge same-event messages. By performing this task at the brokers, Caeva avoids placing this burden on the event subscribers. We also design and implement a distributed aggregator placement algorithm that continuously adapts to message publication patterns with the aim of minimizing the message load within the overlay. We develop an efficient notification scheme for supporting subscriber-specified notification cycles.

In terms of complex event monitoring, current centralized CED techniques have significant limitations that make them ineffective for multi-hop DTN environments. In this dissertation, we present the design and evaluation of a CED planner for Comet, which, to the best our knowledge is the first multi-level CED system in which the multiple DTN nodes share CED tasks. The objective of our planner is to come up with a multi-level plan that minimizes CED costs while respecting a user-specified detection delay tolerance limit. Comet's planner is characterized by its three unique components, namely, push-pull conversion, shorting and shared sub-plan. Push-pull conversion is a two phase heuristic that starts with a simple all-push plan and progressively lowers the cost. First, Comet converts certain carefully chosen push operations to single target pulls; and, when such conversions are no longer possible (due to the delay tolerance), converts remaining push operations to multi-target pull operations where possible. Shorting is designed to counter scenarios in which detecting sub-CEs at every junction does not yield good plans. This module creates virtual CED trees by eliminating junction nodes at various levels of the CED tree. Shared sub-plan is generated by reconstructing CED trees with heuristic model and algorithm. Through extensive experimental evaluation, we have shown that in most cases, Comet produces significantly better plans than existing centralized CED mechanisms.

7.2 Future Work

We propose flexible and scalable framework for event aggregation and redundancy elimination in event monitoring services. The practical impact of the framework is still in its early stage, because

in real world, the decentralized broker overlay network is dynamic. New nodes may continue joining the broker overlay as the new users joining in the event monitoring services. At the same time, some nodes may leave the broker overlay as the users quit the services. Currently, we assume that the broker overlay is relatively stable. Building event monitoring services on a dynamic overlay brings new challenges to the research. As a continuation of this work, we will design new algorithm to adapt to the churn of broker overlay. In a addition to the algorithms, we are also considering using real world traces of Twitter [43] to evaluate the performance of our event aggregation and redundancy elimination in the event monitoring services. Towards this, we have to analyze the content of tweets and build data structure for tweets so that the keywords and associated attributed are extracted and stored.

Complex event detection services will become increasingly more important on emerging delay tolerant networks. This dissertation is the first step towards the goal of building the complex event monitoring services on such overlay networks. To achieve this goal, we have identified several challenges. First, limited resources including storage, power and band width on the DTN nodes. We currently assume each DTN node has unlimited resources. As the resources become limited, the algorithm has to be adjusted to adopt the limitations. Second, load balancing for the nodes in the overlay. Some nodes may host too many sub-plans which results in load imbalance in the overlay. The burst of certain events can also lead to load spikes. A desirable planer for monitoring complex events should balance the load of the overlay as the resources of each node is limited. Third, unpredictable connectivity. Our work is currently based on schedule contact, i.e. the connectivity of the nodes is predictable to some extent. When the connectivity is unpredictable, how to maximize the utility of the connections and minimize the communication cost are the goals we are trying to achieve. Last but not least, real world traces for evaluation. In the future, we are trying to obtain the real world traces by either getting public available traces or building our own. This will help us analyze and improve the system. We foresee that our progress in event monitoring services will provide more insight into different aspects of several real world applications, including design, implementation, analysis and improvement.

BIBLIOGRAPHY

[1] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg, "Content-based Publish-Subscribe over Structured Overlay Networks," in *Proceedings ICDCS*, 2005.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman., "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," in *Proceedings of ICDCS 1999*, 1999.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, 2001.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A Large-Scale and Decentralised Application-level Multicast Infrastructure," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2002.

[5] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, "Content Based Routing with Elvin4," in *Proceedings of AUUG2k*, 2000.

[6] "TIB/Rendezvous," White paper, 1999.

[7] P. Pietzuch and J. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," in *Proceedings DEBS*, 2002.

[8] S. Voulgaris, E. Riviere, A.-M. Kermarrec, and M. van Steen, "Sub-2-Sub: Self-Organizing Content-Based Publish Subscribe for Dynamic Large Scale Collaborative Networks," in *Proceedings of the 5th international workshop on peer-to-peer systems*, Feb 2006.

[9] P. T. P. Felber, R. Guerraoui, and A.-M. Kermarrec, “The Many Faces of Publish/Subscribe,” *ACM Computing Surveys*, vol. 35, no. 2, 2003.

[10] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, “Monitoring streams: A new class of data management applications,” in *Proceedings of VLDB*, 2002.

[11] 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.

[12] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom, “Stream: The stanford stream data manager,” *IEEE Data Engineering Bulletin, 26(1)*, 2003.

[13] Y. Zhou, K. Aberer, and K.-L. Tan, “Toward Massive Query Optimization in Large-Scale Distributed Stream Systems,” in *MIDDLEWARE*, 2008.

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

[15] 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.

[16] S. Seshadri, B. Bamba, B. F. Cooper, V. Kumar, L. Liu, K. Schwan, and G. Zhang, “Grouping distributed stream query services by operator similarity and network locality,” in *SERVICES I*, 2008.

[17] 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.

[18] 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.

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

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

[21] D. Gyllstrom, J. Agrawal, Y. Diao, and N. Immerman, "On supporting kleene closure over event streams," in *Proceedings of ICDE*, 2008.

[22] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *Proceedings of SIGMOD*, 2008.

[23] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM*, 2003.

[24] S. Ratnasamy, P. Francis, S. Shenker, R. Karp, and M. Handley, "A scalable content-addressable network," in *In Proceedings of ACM SIGCOMM*, 2001, pp. 161–172.

[25] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *ACM SIGCOMM*, 2001, pp. 149–160.

[26] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *MIDDLEWARE*, 2001, pp. 329–350.

[27] P. T. Eugster, R. Guerraoui, and C. H. Damm, "On Objects and Events," in *Proceedings of OOPSLA*, 2001.

[28] D. J. Abadi, D. Carney, U. etintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *The Vldb Journal*, vol. 12, pp. 120–139, 2003.

[29] Y. Xing, “Dynamic load distribution in the borealis stream processor,” in *In ICDE*, 2005, pp. 791–802.

[30] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker, “Fault-tolerance in the borealis distributed stream processing system,” in *In SIGMOD*, 2005, pp. 13–24.

[31] J. hyon Hwang, M. Balazinska, E. Rasin, M. Stonebraker, and S. Zdonik, “High-availability algorithms for distributed stream processing,” in *In IEEE ICDE Conference*, 2005, pp. 779–790.

[32] E. Ryvkina, A. S. Maskey, M. Cherniack, and S. Zdonik, “Revision processing in a stream processing engine: A high-level design,” in *Proceedings of the 22nd International Conference on Data Engineering*, 2006.

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

[34] S. Parikh and R. C. Durst, “Disruption tolerant networking demonstration for marine corps condor,” in *Proceedings of MILCOM*, 2005.

[35] 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.

[36] A. Pentland, A. Hassan, and R. Fletcher, “Daknet: Rethinking connectivity in developing nations,” *IEEE Computer*, vol. 37, 2004.

[37] 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.

[38] R. Y. Wang, S. Sobti, N. Garg, E. Ziskind, J. Lai, and A. Krishnamurthy, “Turning the postal system into a generic digital communication mechanism,” in *Proceedings of SIGCOMM*, 2004.

[39] J. LeBrun, C.-N. Chuah, D. Ghosal, and M. Zhang, “Knowledge-based opportunistic forwarding in vehicular wireless ad hoc networks,” in *Proceedings of VTC*, 2005.

[40] 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.

[41] S. Burleigh, V. Cerf, R. Durst, K. Fall, A. Hooke, K. Scott, and H. Weiss, “The interplanetary internet: A communications infrastructure for mars exploration,” in *World Space Congress*, 2002.

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

[43] “Twitter (http://twitter.com).”

[44] J. Chen, L. Ramaswamy, and D. Lowenthal, “Towards efficient event aggregation in a decentralized publish-subscribe system,” in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, ser. DEBS ’09. New York, NY, USA: ACM, 2009, pp. 18:1–18:11. [Online]. Available: http://doi.acm.org/10.1145/1619258.1619283

[45] 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.

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

[47] J. Chen, L. Ramaswamy, and D. K. Lowenthal, “Agele: Dealing with redundant and partial events in a real-world publish-subscribe system,” technical Report UGA-CS-TR-09.001, 2009.

[48] D. Wall, “Mechanisms for Broadcast and Selective Broadcast,” Ph.D. dissertation, Stanford University, 1980.

[49] R. Voigt, R. Barton, and S. Shukla, “A Tool for Configuring Multicast Data Distribution Over Global Networks,” in *Proceedings of INET*, 1995.

[50] L. Song, “A Distributed Algorithm for Graph Center Problem,” Master’s thesis, 2003.

[51] D. Thaler and C. V. Ravishankar, “Distributed Center-Location Algorithms,” *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, 1997.

[52] A. Carzaniga, M. J. Rutherford, and A. L. Wolf, “A Routing Scheme for Content-Based Networking,” in *Proceedings of INFOCOM 2004*, 2004.

[53] J. Chen, L. Ramaswamy, D. K. Lowenthal, and S. Kalyanaraman, “Caeva: A customizable and adaptive event aggregation framework for collaborative broker overlays,” in *Proceedings of CollaborateCom*, 2010.

[54] 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.

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

[56] 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.

[57] M. Bauer and K. Rothermel, “How to Observe Real-World Events through a Distributed World Model,” in *Proceedings of ICPADS*, 2004.

[58] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola, “Epidemic Algorithms for Reliable Content-Based Publish-Subscribe: An Evaluation,” in *Proceedings of ICDCS*, 2004.

[59] L. Fiege, M. Cilia, G. Mühl, and A. P. Buchmann, “Publish-Subscribe Grows Up: Support for Management, Visibility Control, and Heterogeneity,” *IEEE Internet Computing*, vol. 10, no. 1, 2006.

[60] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, “Meghdoot: content-based publish/subscribe over P2P networks,” in *Middleware 2004*, 2004.

[61] Z. Jerzak and C. Fetzer, “Bloom Filter Based Routing for Content-based Publish/Subscribe,” in *Proceedings of DEBS*, 2008.

[62] J. Mocito, J. A. Briones-García, B. Koldehofe, H. Miranda, and L. Rodrigues, “Geographical Distribution of Subscriptions for Content-based Publish/Subscribe in MANETs,” in *Middleware (Companion)*, 2008.

[63] Y. Huang and H. Garcia-Molina, “Publish/subscribe in a mobile environment,” *Wireless Networks*, vol. 10, no. 6, 2004.

[64] R. Baldoni, R. Beraldi, V. Quéma, L. Querzoni, and S. T. Piergiovanni, “TERA: topic-based event routing for peer-to-peer architectures,” in *Proceedings of DEBS*, 2007.

[65] S. Tarkoma, “Dynamic content-based channels: meeting in the middle,” in *Proceedings of DEBS*, 2008.

[66] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, “Constructing scalable overlays for pub-sub with many topics,” in *Proceedings of PODC*, 2007.

[67] R. Lewis, “Advanced Messaging with MSMQ and MQSeries,” 1999.

[68] I. Aekaterinidis and P. Triantafillou, “PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network,” in *ICDCS*, 2006.

[69] B. Chandramouli, J. M. Phillips, and J. Yang, “Value-Based Notification Conditions in Large-Scale Publish/Subscribe Systems,” in *Proceedings of VLDB*, 2007.

[70] G. Cugola and L. Mottola, “A Self-Repairing Tree Overlay Enabling Content-based Routing in Mobile Ad Hoc Networks,” *IEEE Transactions on Mobile Computing*, 2008.

[71] G. Li, S. Hou, and H.-A. Jacobsen, “A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems Based on Modified Binary Decision Diagrams,” in *ICDCS*, 2005.

[72] J. P. Loyall, M. Gillen, and P. Sharma, “QoS Allocation Algorithms for Publish-Subscribe Information Space Middleware,” in *MIDDLEWARE*, 2008.

[73] W. Fenner, M. Rabinovich, K. K. Ramakrishnan, D. Srivastava, and Y. Zhang, “XTreeNet: scalable overlay networks for XML content dissemination and querying,” in *Proceedings WCW*, 2005.

[74] M. Srivatsa and L. Liu, “Securing Publish-Subscribe Overlay Services With EventGuard,” in *Proceedings of ACM-CCS*, 2005.

[75] M. Branson, F. Douglis, B. Fawcett, Z. Liu, A. Riabov, and F. Ye, “CLASP: Collaborating, Autonomous Stream Processing Systems,” in *Proceedings of MIDDLEWARE*, 2007.

[76] B. Chandramouli and J. Yang, “End-to-End Support for Joins in Large-Scale Publish/Subscribe Systems,” in *Proceedings of VLDB*, 2008.

[77] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan, “Resource-Aware Distributed Stream Management Using Dynamic Overlays,” in *ICDCS*, 2005.

[78] N. Jain, M. Dahlin, Y. Zhang, D. Kit, P. Mahajan, and P. Yalagandula, “STAR: Self-Tuning Aggregation for Scalable Monitoring,” in *Proceedings of VLDB*, 2007.

[79] O. Jurca, S. Michel, A. Herrmann, and K. Aberer, “Query Driven Operator Placement for Complex Event Detection over Data Streams,” in *Proceedings of EuroSSC*, 2008.

[80] T. Repantis, X. Gu, and V. Kalogeraki, “Synergy: Sharing-Aware Component Composition for Distributed Stream Processing Systems,” in *Proceedgins of Middleware*, 2006.

[81] R. Adaikkalavan and S. Chakravarthy, “Events must be complete in event processing!” in *Proceedings of ACM-SAC*, 2008.

[82] A. Lindgren, A. Doria, and O. Schelen, “Probabilistic routing in intermittently connected networks,” in *SIGMOBILE Mobile Computing and Communication Review*, 2004, p. 2003.

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

[84] K. Scott and S. Burleigh, “Bundle protocol specification,” *IETF RFC 5050, experimental, November 2007*.

[85] W. Ivancic, W. Eddy, L. Wood, J. Northam, and C. Jackson, “Experience with delay-tolerant networking from orbit,” *preprint for the International Journal of Satellite Communications and Networking, special issue for best papers of ASMS 2008*, 2010.

[86] L. Wood, W. Ivancic, W. Eddy, D. Stewart, J. Northam, C. Jackson, and A. da Silva Curiel, “Use of the delay-tolerant networking bundle protocol from space,” in *59th International Astronautical Congress*, 2008.

[87] W. Ivancic, W. Eddy, L. Wood, D. Stewart, C. Jackson, J. Northam, and A. da Silva Curiel, “Delay/disruption-tolerant network testing using a leo satellite,” in *Eighth Annual NASA Earth Science Technology Conference (ESTC 2008)*.

[88] W. Zhao, M. Ammar, and E. Zegura, “A message ferrying approach for data delivery in sparse mobile ad hoc networks,” in *ACM MobiHoc*, 2004.

[89] P. Yang and M. Chuah, “Efficient interdomain multicast delivery in disruption tolerant networks,” in *Proceedings of MSN*, 2008.

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

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

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

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