Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2440

Dynamic Operator Splitting in Mobile CEP Scenarios

Stefan Schmidhäuser

Course of Study:	Informatik
Examiner:	Prof. Dr. Rothermel
Supervisor:	Dipl.-Inf. Beate Ottenwälder
Commenced:	30. September 2013
Completed:	1. April 2014
CR-Classification:	C.2.1, C.2.2, C.2.4

Abstract

With an increasing number of mobile devices being used in modern day to day life, applications using data provided by their environment are becoming more and more common. Complex Event Processing systems are a popular approach to realize them. Usually the data, collected for example by the mobile devices' built in sensors like GPS is sent to a stationary host for evaluation. Since distances between host and data source might be quite large, this is usually done by using communication standards like 3G or GSM. Since energy costs of such messages are rather high it is in some cases more energy efficient to evaluate some data directly on the mobile device before sending it, in order to reduce communication. In large scale networks however, this will still result in a lot of expensive 3G or GSM messages being sent, since each mobile device will have to send its data eventually.

This work will introduce algorithms to create a simple network structure that takes advantage of the mobile device's locality in order to create clusters of mobile devices. In these clusters, members use the much cheaper wireless lan communication to collect data, which will then be pre-processed locally before being sent to the host via the more expensive 3G or GSM for final processing. To make pre-processing possible we will also examine Operators used in CEP systems in terms of their ability to be distributed further, and provide a classification based on this.

Finally we will evaluate our system by performing simulations. We will test the general performance of our algorithms by measuring the energy consumed to transmit events. Our evaluation shows that our approach is able to safe energy for systems that produce events with a high frequency.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Introduction

Mobile devices are very common and widespread these days. These devices are usually equipped with a number of sensors that provide data which can be used in any number of applications. A rather simple example for this would be a navigation application, which will use the device's GPS data to calculate routes for a user. Utilizing the communication ability most devices possess, this data can be shared. This will allow applications to not only use sensor data of the device they are running on, but instead to use data provided by a large number of sensors, which do not necessarily have to be located close to the application's device. For example a traveler might want to look up the temperature at his destination, in order to dress accordingly.

One way to realize more complex applications with such sensor data, is using Complex Event Processing (CEP). A CEP system collects events from sources. Sources can be any kind of sensor or device that provides data in form of events for the CEP system. These events are then scanned for patterns by an operator running on a device that collects said events. When an operator detects a certain pattern in events, it uses a correlation function on them. This function uses the information provided by the selected events to create a new complex event, that describes information on a higher level of abstraction. Three employees entering a room shortly after one another before any of them leave are separate events, but by putting them together we can get the complex event "meeting". Complex events can be used by other operators for further correlation. Alternatively complex events can be consumed by consumers, which usually are applications that use these complex events to complete their task.

In these systems events are typically produced by a large number of sources and collected centrally by devices that host operators, for example servers in data centers. This means that all events will have to be transmitted by the sensor device, which is usually done by using 3G. Sources that produce events with a high frequency can put considerable strain on the source device's energy resources, for example battery life of a mobile phone.

In fact if sensor data has to be transmitted at a high frequency, computation cost is small enough and computation on the sensor device reduces the need for communication enough it can be energetically more efficient to run parts of the application on the sensor device itself [Vet12]. This shows that communication cost may not be underestimated and may provide room for improvement.

In this work we will reduce communication cost by grouping sensor devices into localized clusters and, instead of using 3G, using the cheaper WLAN communication to calculate partial solutions for those clusters, before sending them to the final application device. Of course this will not be possible for all applications, which is why we will first classify applications in regard to this approach. Then we will provide algorithms to build a simple cluster network based on WLAN communication, and evaluate them via simulation.

Structure

This Document is structured in the following way:

Chapter 2 – Background: This chapter will provide basic information about the preliminaries of this work.

Chapter 3 – System Model: Here we will introduce the system model we use as a foundation for this work.

Chapter 4 – Problem Description: Chapter 4 will describe the focus of this work in detail.

Chapter 5 – Operator Classification: This chapter will discuss conditions for viability of operators in context to the provided algorithm and provide a classification regarding that criteria.

Chapter 6 – Algorithmic Solution: Chapter 6 will introduce and explain the algorithm used in this work in detail.

Chapter 7 – Algorithmic Solution: Chapter 7 will present evaluation results obtained by simulations of an implementation of our algorithm.

Chapter 8 – Related Work: In this part we will introduce other works related to CEP that bare relevance to this work.

Chapter 9 – Conclusion: Chapter 9 will summarize our work and propose approaches for future work.

2 Background

This chapter will introduce the basic foundations of this work. We will start by taking a closer look at the Complex Event Processing (CEP) paradigm. Afterwards we will go over the different communication technologies used by mobile devices, mainly 3G, GSM and WLAN. The last part of this chapter will give insight in the energy model used to estimate energy costs.

2.1 Complex Event Processing

The Complex Event Processing paradigm first surfaced in the mid 1990s. Stanford University's Rapide Project[LV95] is credited to be the first CEP model. CEP Systems have since then become more and more relevant since they allow to process obtained data in real time, which is interesting for many applications, like for example real-time market analysis on the stock exchange. Due to their properties CEP systems are also well suited for use in distributed systems, which allows for high scalability.

The basic idea behind a CEP system is the collection of data provided by information sources in the form of events. An event may be any piece of information, like changes in a company's stock value. The collected data provided by different sources is then interpreted by the system. This is accomplished by checking the incoming events for patterns or if they meet certain requirements, and if so correlating them in order to form Complex Events. These Complex Events may then again be used as data for other parts of the system or consumed in an application.

In accordance with the work 'Moving Range Queries in Distributed Complex Event Processing' [KORR12], the detection process in the Complex Event Processing system is realized by the following three components.

Source: Entities that produce information in form of atomic events are sources. A thermometer notifying the system about a change in temperature would be an example for a source.

Operator: Operators collect atomic or complex events from sources and other operators. The collected events are then interpreted and possibly correlated into new complex events. These new complex events can then be sent to either operators for further correlation or to consumers. An example would be an operator that collects the temperature of several locations and calculates the mean. We will give a more detailed description of the operator model in Chapter 5 as part of the classification process.

Consumer: Consumers are the systems end points. They receive events from operators and consume them to fulfil a certain task. An example for a consumer might be a weather application that visualizes mean temperatures of several areas on a map.

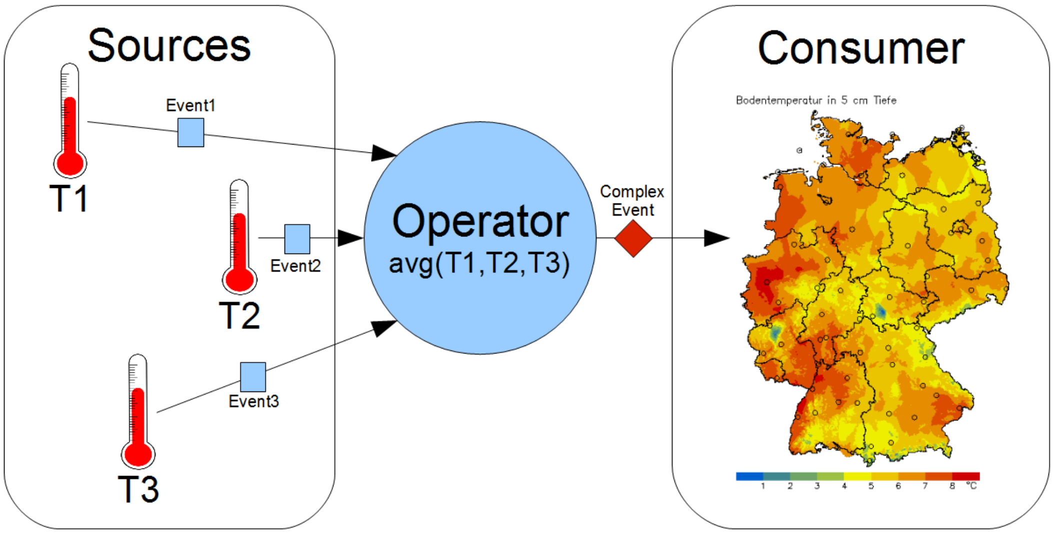


Figure 2.1: Illustration of the example CEP system used to describe the components.

Figure 2.1 illustrates the example used in the description of the components. T1, T2 and T3 are sources that measure temperature at their location and submit their readings to the operator. The operator calculates the average temperature of the area and informs the consumer via complex event message. The consumer uses the data to visualize the results.

Operators often implement complex correlation functions. Many of these complex operators can be split into a number of sub-operators, that perform certain steps of the correlation separately and provide their parent operator with the results. By consecutively splitting up sub-operators, if possible, we are able to create operator-trees. The leaf nodes of such an operator-tree are basic operators that perform simple correlations on events they collect directly from the sources. Operators placed further up in the tree typically receive their incoming events from lower level operators and are becoming more complex towards the root. The root operator itself will produce the same correlations as the original unsplit operator.

The advantage of creating operator-trees is that the different operators in the tree can be distributed among different nodes in an infrastructure, which allows the system to perform simpler correlations early on and more complex ones while the events make their way through the infrastructure towards their goal. This helps to reduce network overhead and helps when implementing distributed CEP systems.

[KORR12] [OKRR13] [KKR10] [RDR10]

2.2 Communication between mobile devices

Modern mobile devices have various ways of communicating with stationary servers or other mobile devices. In this work we will focus on the most common communication interfaces, namely Wireless LAN, GSM and 3G.

Wireless LAN uses radio waves in order to communicate with other devices. Frequencies used for communication are typically based on the IEEE 802.11 standard. With WLAN it is possible to connect to an access point, which bridges the WLAN device to another network, such as the internet. More interesting for us however, is the possibility to communicate directly with other WLAN devices via peer-to-peer connection. Peer-to-peer connections may also be used to establish an ad-hoc network, in which devices that would otherwise be to far apart for communication use other network participants as relay points. WLAN communication is very fast while consuming less energy for transmissions than other standards, making it rather energy efficient. Its range however is limited with a maximum distance of about 100 meters.

Global System for Mobile Communications (GSM) describes the second generation of communication standards used in mobile phones. These standards include General Packet Radio Service (GPRS) and Enhanced Data rates for GSM Evolution (EDGE). Both standards are packet based an TCP-IP compatible. A GPRS service is assigned up to eight frequencies for transmission by the GSM network, over which it can transfer data in form of packets. Since several channels can be used at once, transmission speeds are higher than older standards. Frequencies are shared among users, and only used when data actually has to be transferred. EDGE is a superset of GPRS, using different coding schemes and modulation, and can be deployed on any GPRS capable network. It is able to handle about 4 times as much traffic as GPRS. Since GSM uses the cellular network to transmit data, its range is very high.

3G stands for third generation and is actually not a single service. 3G is a collection of communication standards that comply with the International Mobile Telecommunications - 2000 specifications. Standards included in 3G are among others UMTS and CDMA2000. These are perhaps the currently most appropriate standards used in mobile communication. Both are based on spread spectrum technologies and allow multiple transmitters to send on shared frequencies. Services based on these standards provide faster data transmission than the older GSM. 3G transmissions are using the cellular network, which means their range is very high.

[Sch03]

2.3 Energy in mobile device communication

One point that always has to be considered when working with mobile devices is always the limited availability of energy. Mobile devices are typically powered by accumulators that can only store a limited amount of energy. Since we want to stretch battery life as much as possible we want to optimize applications running on the device due to energy consumption. According to 'An Analysis of Power Consumption in a Smart-phone' [CH10] the majority of power in modern mobile phones is drained by Communication services, CPU and the graphics hardware.

Interesting for this work is primarily the energy consumed by communication services, which is discussed in more detail in 'Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications'[BBV09].

One of the results is that most of the energy consumed for data transmission using GSM and 3G standards isn't used for the transmission itself. In order to transmit data 3G and GSM have to enter a *high-power* state. For 3G this already uses up a considerable amount of energy. After transmitting the data both standards remain in this *high-power* state for a short duration, using up energy without actually transmitting data. [BBV09] refers to this as *Tail-Energy*. For message sizes of 50Kb, *Tail-energy* actually makes up over half the power used when transmitting using 3G and a third when using GSM. Taking the *Tail-Energy* in consideration it is favorable to transmit data with a high frequency or in preferably large files.

In general GSM uses less energy than 3G until a certain data size, at which point the slower tranfer speeds result in increased time needs.

WLAN does function without a *high power* state and only uses energy when actually transmitting data. However if connection to an access point in an infrastructure, WLAN will have to perform a *Scan Association* to synchronize with the device. Since this is a one time operation and the device remains connected this should be irrelevant most of the time. WLAN outperforms the other standards especially for bigger data files. Due to its high transmission speeds WLAN is able to send larger amounts of data in less time than the other standards. Because WLAN already uses less energy for transmission even if the same time is needed, it outperforms the other standards with growing file size.

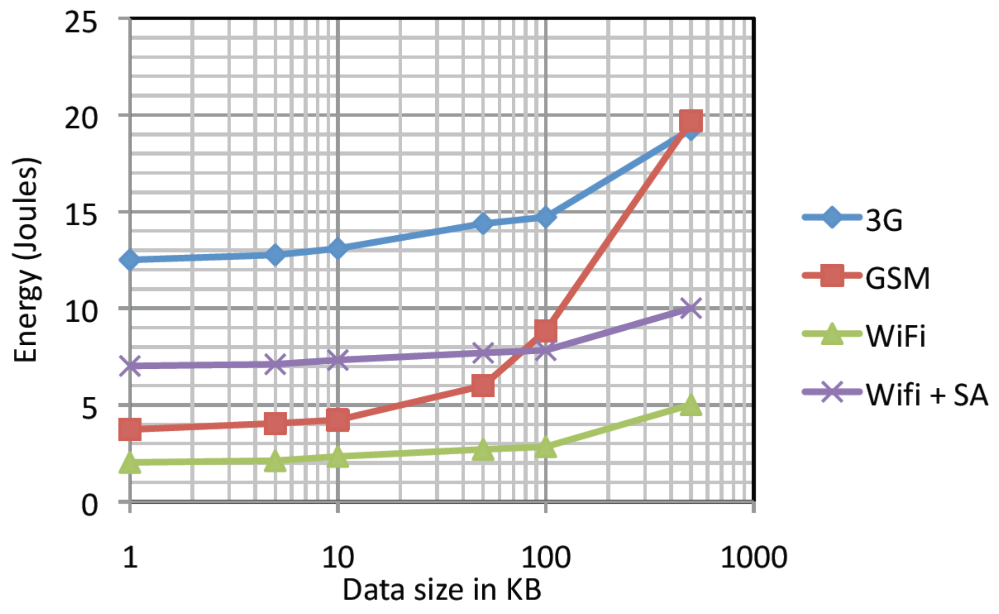


Figure 2.2: WiFi versus 3G vs GSM measurements: Average energy consumed for downloading data of different sizes in a 20 second interval. [BBV09]

Figure 2.2 shows the average energy needed for transmissions in a 20 second interval with growing file size. Energy costs for GSM grow fast due to its low transmission speed, which means that for larger files the time needed to transmit the file is long compared to the other standards. WiFi + SA assumes that the device needs to perform a scan-association process before every single transmission, while WiFi shows the actual transmission cost when already connected to an infrastructure.

3 System Model

This chapter will describe the system model used in this work and explains the assumptions we make about it.

3.1 Infrastructure model

Figure 3.1 illustrates the physical layout of our system. The system consists of two types of components, a server and a set of mobile devices.

Mobile devices are each equipped with their own CPU and internal data storage, making them able to execute operators. As the name implies these devices are considered mobile, meaning their position will change over time. Since the devices are moving a permanent power supply is unrealistic. Instead power on these devices is provided by a limited power source like an accumulator. Every device also possesses at least one sensor for collecting event data. Mobile devices typically posses GPS sensors, however we cannot guarantee that GPS is activated on all devices, and hence will not expect it. As for communication abilities, mobile devices are able to transmit and receive data via 3G, GSM and Wireless Lan. 3G and GSM communication is used to communicate with the server, whose address has to be known at all time. Because of the long range of 3G and GSM communication the server is always considered to be in range. Wireless Lan is used to communicate between mobile devices. For addressing purposes each mobile device has its own unique identifier, like for example a Mac-address. Since Wireless Lan communication range is limited, other devices will only be reachable when they are physically close. Figure 3.1 depicts 3G and GSM communication as solid lines and Wireless Lan as dotted lines. Due to their hardware capabilities mobile devices can take the role of consumer, operator and source in the CEP system.

The Server in our model has its own CPU and internal data storage. It is able to communicate with mobile devices via 3G and GSM. Power is supplied by a limitless power source to guarantee high availability of the server. The server is an entry point to an infrastructure that hosts the CEP system's operators. As such it collects all events produced by sources in a certain area. Because of that it will have to know the IDs of all mobile devices that supply the operator with events. It then supplies the nearest infrastructure member, hosting a leaf operator with the collected events. This can but does not have to be the server itself. In our system the server can take the roles of a consumer and operator.

In a typical CEP system operators are usually executed on servers in an infrastructure, whose entry point will be the server in our scenario. Mobile devices will act as sources, which transmit atomic events to the first server that is part of the infrastructure, but will

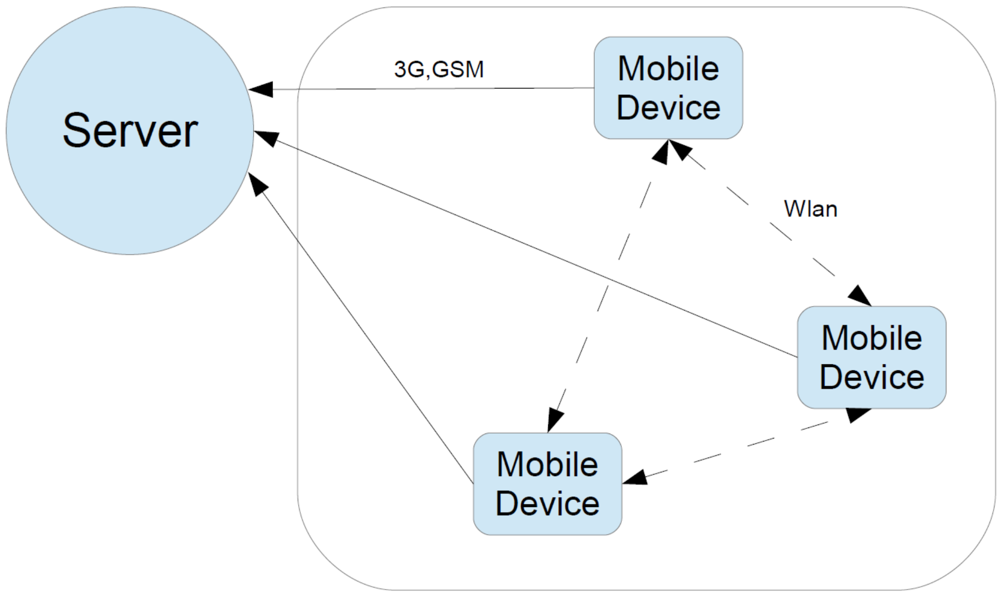


Figure 3.1: The system model.

also execute operators. Once the events are transmitted to the infrastructure, events are routed to the corresponding operator-graphs. Event distribution can be performed by using publish/subscribe [TKK+10].

If a mobile device first joins the CEP system, it registers with the nearest. At this point the mobile device also receives all data necessary for future operations.

As additional assumption we consider the Wireless Lan communication range to be constant. In reality this is rarely the case, since any kind of physical obstacle interferes with the signal.

3.2 Operator model

As mentioned in Chapter 2 we use an operator model similar to the one introduced in [KORR12]. Operators receive events in order over incoming event-streams. Each source supplies events to its own specific event-stream. As a result the operator will have as many incoming event streams as it has sources. Events are kept in memory on the server for as long as they are considered relevant.
Correlation detection is realized using a correlation function on a selection of events. This selection is determined by using so-called selection windows, which scan segments of an event-stream for events that qualify for correlation. Each event-stream possesses its own selection window, with a specific number of *slots*. *Slots* indicate how many events of the particular event-stream are necessary to perform the correlation. Once a selection window contains enough events to fill all its *slots*, the event stream is marked as correlation ready. In

order to start the correlation a certain number of event streams have to be in the ready state. The necessary number of ready event-streams may vary for different types of operators. In order to calculate the mean temperature of an area for example, all event-streams will have to be ready at the same time, since we will need temperature readings from all devices to calculate the correct mean. If however we want to find the maximum temperature in the area we can start filtering the results on any number of event streams being ready, because correlation on a subset of event-streams will not alter the final result. The actual number of streams necessary for correlation has to be defined by the programmer.

The correlation step itself is done by applying the correlation function on the selected event-streams. This function is implemented in the operator and uses the information provided by the incoming events to create one or more new complex event.

After correlation is finished, the incoming events are processed according to the selected consumption policy. Usually they are either deleted or kept in storage for further use. Finally selection windows are repositioned on their respective stream, following a previously defined selection policy that defines a window's shift on its event-stream after correlation.

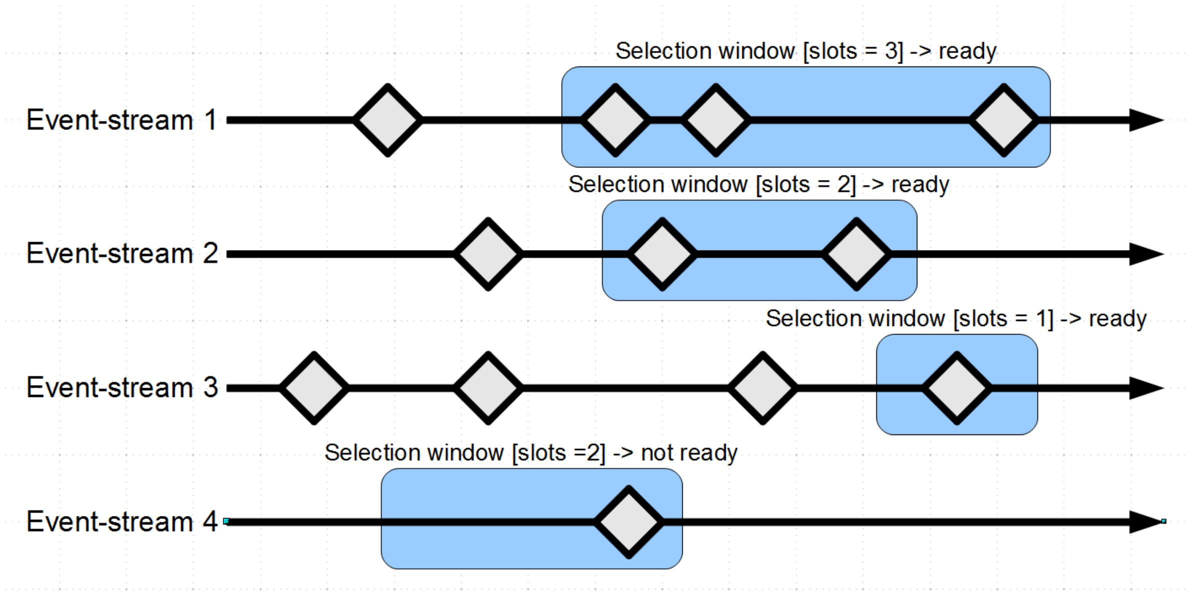

Selection window [slots = 3] -> ready
Event-stream 1
Selection window [slots = 2] -> ready
Event-stream 2
Selection window [slots = 1] -> ready
Event-stream 3
Selection window [slots =2] -> not ready
Event-stream 4

Figure 3.2: Snapshot of an operator state.

Figure 3.2 shows a snapshot of a possible event-stream configuration of an operator. Event streams are displayed as lines, while the rhombuses represent events. Selection windows are illustrated in blue. In the case at hand, selection windows of streams 1 and 3 are in the ready state. In order to create a new correlated event the operator will also need the selection windows of stream 2 and 4 to fill all the required slots.

3.3 Energy model

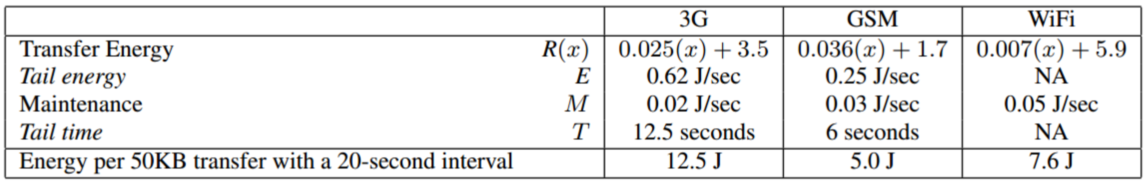
To approximate energy cost of data transfer in our system we will use the experimental results from 'Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications' [BBV09] as a baseline.

3G and GSM communication may cause additional ramp an tail energy costs besides the actual transfer cost. This is the case because both methods need to enter a high-power state before sending. This state will be kept for 6 seconds in case of GSM or 12 seconds for 3G. If in this time no further transfer occurs the system will power down again and all energy used to keep the high-power state up is wasted. For highly frequent transfers however, the high-power state is maintained continuously and the upkeep energy is not wasted since we save the ramp cost to enter it. As a result we will approximate energy cost of communication with a high frequency by just looking at actual transfer costs. If the message frequency is low enough that a protocol will return to its low-power state we will add ramp and tail costs to the transfer energy according to the results in [BBV09].

For Wireless Lan we assume that communication between mobile devices in range of each other is possible without building an Ad-Hoc network. In this case we can use the experimental results for wireless transmission without scan-assertion also from [BBV09].

However all results for GSM and Wireless Lan communication are for download processes. In order to approximate upload cost we use experimental values from 'Context-for-wireless: context-sensitive energy-efficient wireless data transfer' [RZ07]. These results show that GSM upload transfer cost is approximately double the download cost. For Wireless Lan uploads, transfers costs about 125 percent of download cost.

This seems to fit the results for upload transfer costs for 3G in [BBV09] well, that are also approximately double as high as the download transfer cost. However since Ramp and Tail costs are the same the overall costs for sending a message are not quite twice as high for transmissions with a low frequency.



		3G	GSM	WiFi
Transfer Energy	$R(x)$	$0.025(x) + 3.5$	$0.036(x) + 1.7$	$0.007(x) + 5.9$
Tail energy	E	0.62 J/sec	0.25 J/sec	NA
Maintenance	M	0.02 J/sec	0.03 J/sec	0.05 J/sec
Tail time	T	12.5 seconds	6 seconds	NA
Energy per 50KB transfer with a 20-second interval		12.5 J	5.0 J	7.6 J

Figure 3.3: Formulas to estimate energy cost of downloading data. [BBV09]

Additionally [BBV09] provides a number of formulas to estimate the energy costs of downloading data, which are given in Figure 3.3. *Tranfer Energy* is the amount of energy actually needed to transfer the data. When using the formulas for WiFi we have to consider that the additional transfer cost of 5,9 Joules represents the scan-association process, which will not be necessary if a device is already connected to a structure. The addends 3.5 and 1.7 Joule for 3G and GSM stand for the ramp cost to get the communication channel into its high power state. *Tail time* gives the time the channels are kept in a high power state after transmission. Hence when transmitting data again over the same channel before the respective *Tail time* has passed, this additional cost will also not apply. *Tail energy* is the cost of keeping the channel in a high power state for one second, while *Maintenance* is the cost of maintaining channels in their low power state.

Protocol	**Action**	**Energy (Joule)**
WLan	receive	1.25
GSM	receive	3.5
3G	receive	3
WLan	send	1.5
GSM	send	7
3G	send	6

Table 3.1: Energy consumption of a 50kb message approximated in high-frequency mode

Table 3.1 shows the approximated cost values for sending and receiving messages with a size of 50kb in a high-frequency mode, which ignores tail energy. Table 3.2 also shows cost values for messages of 50kb of data, but in a low-frequency mode where tail and ramp energy are included.

Protocol	**Action**	**Energy (Joule)**
WLan	receive	1.25
GSM	receive	6
3G	receive	13
WLan	send	1.5
GSM	send	10
3G	send	17

Table 3.2: Energy consumption of a 50kb message approximated in low-frequency mode

4 Problem Description

A CEP system consists of sources, operators and consumers. Sources provide operators with event data which in turn is used to produce correlated events. These correlated events are then used by consumers to complete their task.
In current CEP systems it is common to place operators of an operator tree on Servers in an infrastructure. This has the advantage that Servers usually have sufficient processing power and energy resources to handle even the most complex operators. Since operators typically work on events from several sources other than the server hosting the operator itself, this means that events will have to be transmitted to this server. Typically events from sources are first collected by what we will call entry servers to the infrastructure. An example for such an entry server could be a server that collects all events produced by sources located in a specific area. These entry servers either supply a server hosting a leaf operator of the operator tree or are hosting such an operator themselves.

Mobile devices like smart phones are becoming more popular and widely spread. Most of these devices are equipped with a number of sensors like GPS. It is not surprising that new applications based on CEP systems use them as sources. When considering the ordinary CEP system model, a large number of mobile devices used as sources will cause a lot of event messages that have to be sent to the infrastructure. This will result in considerable network load on the entry servers.

Mobile devices mostly realize event transmission to entry servers by using 3G or GSM communication as shown in Figure 4.1a. Since reporting events continuously drains the limited power supply we want to keep it as cheap as possible. Considering the lower energy cost of Wireless Lan compared to 3G and GSM it seems to be a desirable alternative. However the range of Wireless Lan communication is very limited, which normally makes the entry server unreachable. So direct event transmission via Wireless Lan will rarely be possible.

Another approach to reduce energy drain on mobile devices is to simply place the operator on the mobile device itself and having it stem the atomic events, in hopes to reduce communication cost to compensate for the additional computational cost and ideally even safe energy overall. [Vet12] states that this is indeed the case for operators with low computational cost, few incoming event-streams and high correlation frequency.

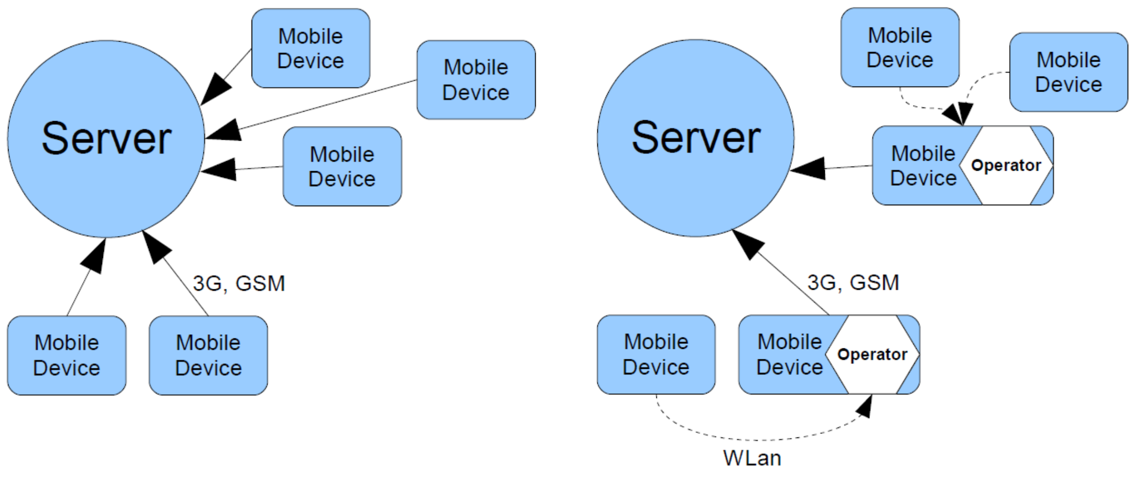
While the entry server typically is out of range for Wireless Lan communication, other mobile devices acting as sources might actually be close enough. In this case, sending events to other mobile devices is cheaper than sending events to the entry server. In order to reduce overall energy consumption it seems sensible to take advantage of the cheaper Wireless Lan transfers and determine which devices are in range of each other. We can then collect events from these devices locally and assign only one device to send the events to the entry server collectively, using the more expensive 3G or GSM.

While this effectively reduces the number of expensive 3G and GSM messages we have to keep in mind that mobile device are moving. This means that devices will eventually move out of or into Wireless Lan range. As a result the number of devices from which events can be collected will be changing over time and has to be redetermined regularly.

Relaying all network traffic over one mobile device will increase the energy drain of that particular device. Instead of sending only its own event data it now has to send more data, depending on how many sources it relays the increase can be rather high. Since sending larger messages results in up to exponentially more energy cost per transfer we want to keep the messages to the server as small and low frequent as possible. To achieve this we could incorporate the approach of executing an operator directly on the mobile device. By creating new correlated events before sending data to the server we can potentially reduce message size enough to reduce overall energy consumption on the relaying device. This approach is illustrated in Figure 4.1b.

That said, collecting events before transmitting them to the server will cause fundamental problems regarding the operator. Correlation detection in operators require selection windows of a defined set of event-streams to be in ready state. By sending collected event data we reduce the number of event-streams of the operator, and in case of *early correlation* probably even the type of event received by event-streams. In order to keep the operator on the server operational we will have to account for this if possible.

Figure 4.1 shows the standard approach and a system with relayed events. 3G and GSM transfers are illustrated as straight lines. Wireless Lan communication is depicted as dotted lines.



(a) Standard event transmission **(b)** Relayed event transmission

Figure 4.1: Event transmission models

The problem addressed in this thesis is the minimization of the energy consumption of a CEP system caused by transmitting events from sources to entry servers of the infrastructure. Any event transmission within the infrastructure is not part of this work.

5 Operator Classification

We want to realize *relayed event transmission*. That means that instead of sending events directly to an entry server, mobile devices acting as sources will look for other devices in WLAN range. If a group of mobile devices is in range of each other, one of them will collect all events produced by that group, using WLAN transmissions. Afterwards this device will send the collected events to the entry server in one transmission. Since collecting events can result in larger amounts of data to be sent, we also want to perform *early correlation* on this mobile device. This means that the mobile device will execute an operator of some sort on the events to reduce the amount of data. *Relayed event transmission* and *early correlation* require operators to be adaptable to a certain degree. Different operators can be adapted more easily to the new system than others. In this chapter we will take a closer look at the challenges of our approach and how compatible different classes of operators are with *early correlation*.

5.1 Requirements when using relayed transmission and early correlation

In this section we describe the requirements of an operator in order to be used in a system with *early correlation* and relayed communication. The more this properties are met the better it will work with our system. We will now describe these two properties.

Working on a varying number of incoming event streams: In a typical CEP system each mobile device would send its event messages directly to the entry server, who will forward it to the nearest leaf operator in the infrastructure. According to the operator model we use in this work, that operator will have an incoming event stream for every source or in our case mobile device. Using *relayed event transmission*, we collect events from mobile devices that are in range of each other. These events are then sent to the server by only one of these devices. Instead of receiving the collected events over multiple event-streams, it will only receive events on the stream assigned to the device that collected them. Since correlation will only be performed when a certain number of event streams are in the ready state this might cause problems. Without modification the operator might expect some of the collected events to arrive on their original source's event-streams, which are now empty, before being able to initiate the correlation. On top of that mobile devices are moving and will be in range of different mobile devices over time. This means that the groups, which are collecting events will also change. If the size of such a group changes, a different number of the operator's event-streams is not being served events. A change in member devices of the group on the other hand, will cause that a different event-stream than before will not receive events. To keep the operator functioning it needs to be able to work on a varying number of event streams or we will have to find an alternative solution to supply the unserved event-streams.

Processing new types of complex events Since we are using *early correlation* we are executing an operator on the mobile device collecting the events. We do so in order to keep the amount of data the device will have to transmit as small as possible. By correlating the atomic events early we will probably create a new type of complex-event, that is sent to the entry server. This could cause problems for the operator since each event-stream typically only receives a predefined type of events. In our case the operator's events-stream would receive the new type of complex-event instead. Without modification the operator will not be able to process this new type of complex event. This starts with event selection on the operator's event-streams. Selection windows scanning the incoming events will not know how to handle the complex event, and hence the event-stream might not be able to enter the ready state. Even if the selection windows accept the new complex event we will probably still need to modify the correlation function itself. The preprocessed event data contained in the complex event could be presented in a format unknown to the unmodified correlation function. In this case the correlation function will not be able to process the complex event. To implement these modifications we will need extensive knowledge about the operator, which has to be provided by the programmer.

5.2 Operator Classification

We have introduced the main criteria determining how well operators are suited to be used with our altered CEP system. The next step is to provide a classification based on suitability.

Fixed Operators: There are two cases in which operators are part of this class. If we have absolutely no knowledge of the operator and are thus unable to adapt it to better suit our system or if we do have extensive knowledge about the operator, but the operator itself simply cannot be modified. In this case the only way to make *relayed event transmission* possible is to collect events of mobile devices using Wireless Lan when devices are in range and save them as (event,source) pairs. These pairs can then collectively be sent to the server. The server will then have to look through every (event,source) pair and distribute the events to the operators event-streams as if they were received by the source device.

Figure 5.1 illustrates this case. Modified components are shown in red. Mobile device 2 collects events from mobile devices 1 and 3 and builds (event,source) pairs. Mobile device 2 then sends all collected pairs including its own to the server, where the *Distribution Module* that we have to implement distributes them to their respective event streams. Since Mobile device 4 send its event data normally it can simply be forwarded to its event stream.

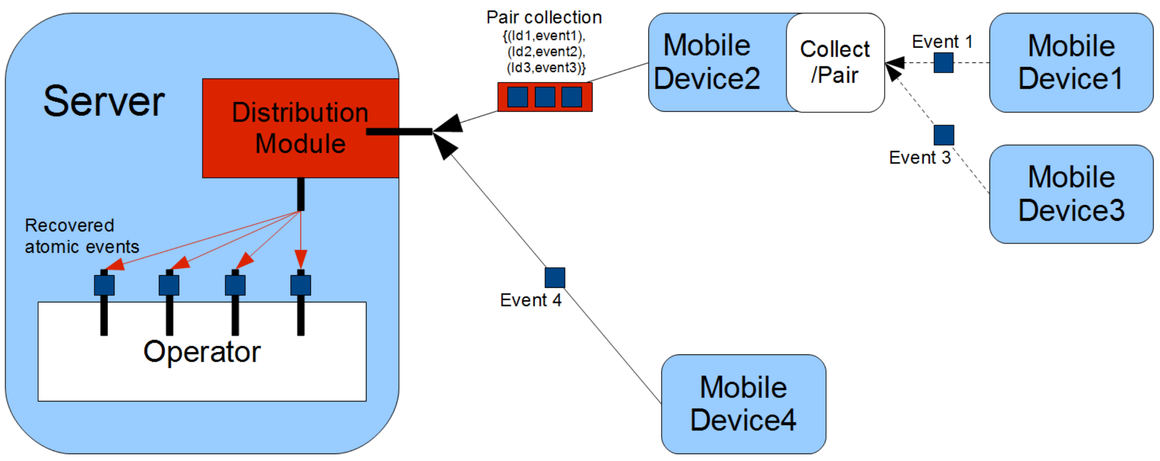


Figure 5.1: Relayed event transmission for Fixed Operators.

Modifiable Operators: Operators fall into this class if we do have extensive knowledge of the operator, which has to be provided by the programmer, and are able to adapt it to do the following things: work on a varying number of event-streams and process the new correlated event type produced by *early correlation*. Instead of simply collecting events and send them collectively as a concatenation we are in this case able to pre-process some of the information given by the events. In doing so we can minimize the size of the transfer to the server to safe energy on the relaying device. To deal with the varying number of event streams we can either alter the operator itself or if possible use our knowledge of the operator to feed the unserved event-streams proxy events created by the server from the received correlated event.

An example for this kind of operator would be the average temperature in a weather application. Usually each source would send their measured temperature to the server. The server would then calculate the sum of all temperatures and divide by the number of sources or event streams. However with relayed transmission sources could report their measurements to a relay device, which would calculate the temporary sum. Simply sending this temporary sum to the server would falsify the overall result. Since there would be less event-streams in the operator receiving data, the number it would divide the final sum by would be too low. So we either have to feed the unserved event streams with temperatures of zero, or add a weight to the correlated event sent by the relay device. In the first case the operator would divide by the correct number of sources or given a weight it could determine the correct number from the correlated event.

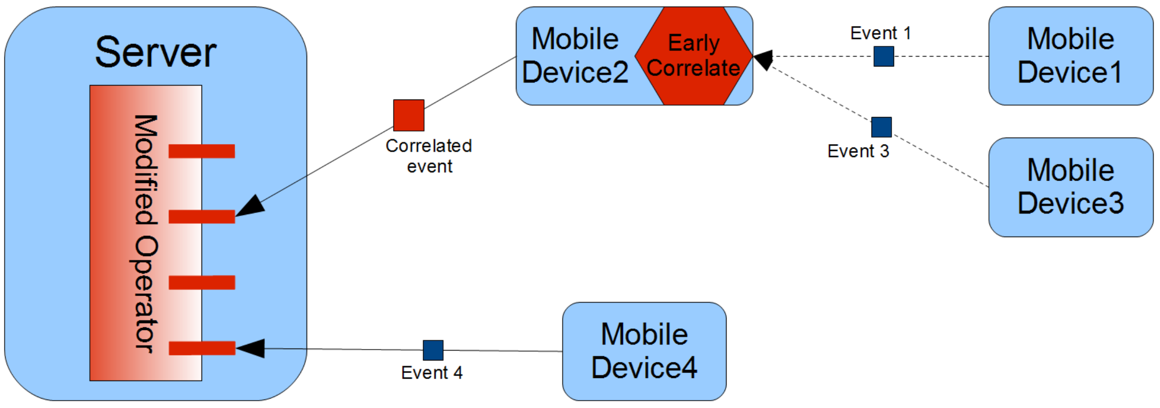


Figure 5.2: Relayed event transmission for Modifiable Operators.

Figure 5.2 shows an example for *relayed event transmission* for *Modifiable Operators*. Modified components are illustrated in red. Mobile device 2 collects events from mobile devices 1 and 3 and performs *early correlation* on all collected events, including its own. Mobile device 2 then sends the correlated event to the server. The operator is modified to also accept this new type of event and work with a variable number of event streams. Since Mobile device 4 send its event data normally it can simply be forwarded to its event stream.

Filter Operators: Operators in this class have two properties. The events produced by the operator are of the same type as those it receives from the sources. In addition the operator produces the same result indifferent of how often it is applied and in which subsets it receives the mobile device's events. This case is fairly easy to implement. We can simply place a copy of the operator on the relaying device and transfer the result to the server. Since outgoing and incoming events are of the same type, message size equal to the size of a single event from a source, which is ideal for our system.

Considering our weather application, we can name the maximal temperature as an example. The operator simply scans all incoming events and selects the one with the maximal temperature. When using relayed transmission, we simply have to the same on all locally collected events. As long as all mobile devices send their events to the server in some ways, the operator will always produce the correct result.

The system shown in Figure 5.3 is a simple example for this scenario. Mobile device 2 collects events from mobile devices 1 and 3. It then executes a copy of the operator on all collected events, including its own and sends the result to the server. The operator does not have to be modified in this case. Since Mobile device 4 send its event data normally it can simply be forwarded to its event stream.

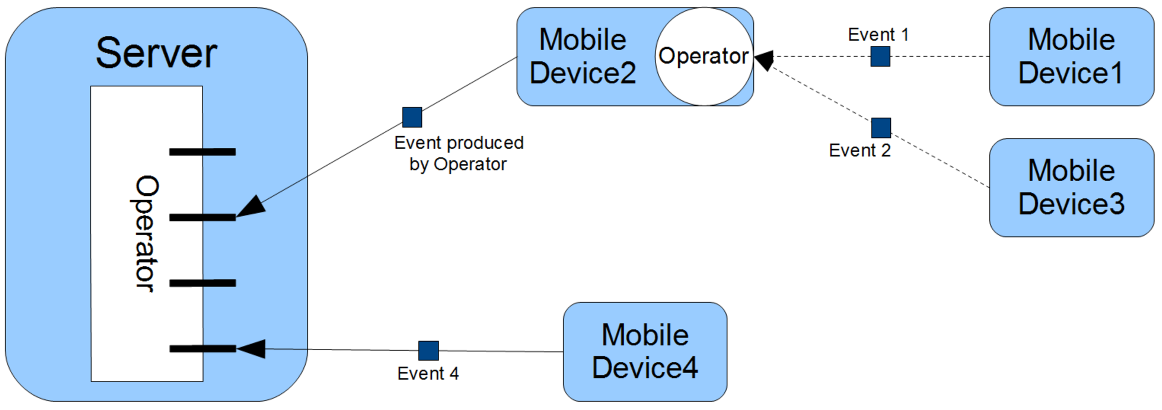


Figure 5.3: Relayed event transmission for Filter Operators.

6 Algorithmic Solution

We will now introduce the algorithm we use to implement *relayed event transmission* and *early correlation*. Our goal is to relay messages over mobile devices that are able to communicate using Wireless Lan in order to reduce the number of more expensive transmissions using 3G or GSM. The relaying device will collect events via Wireless Lan and then send them collectively. To keep the amount of data that has to be transmitted small we will perform some sort of *early correlation* on the relay device. *Early correlation* poses special challenges to operators used in CEP system, since it might create new complex-events the operators are not able to process. Hence we might need to modify their event-streams and correlation function to be able to handle this new type of complex-event. These modifications are performed according to Chapter 5.

Section 6.1 will explain the general idea behind the algorithm. The following sections will then explain how we implemented the different aspects of our algorithm in detail. Section 6.2 will describe the initialization process of the system in more detail. Section 6.3 will explain the algorithm we use to find groups of mobile devices that are close enough to each other to use *relayed event transmission* and *early correlation*, while Section 6.4 will discuss the algorithms we use to maintain these groups over time.

6.1 Algorithm in general

During the initialization stage of our algorithm, the mobile devices will make themselves known to the server. Each device will receive the server's address to which it can send events using 3G or GSM at all times. Additionally every device will receive a copy of the operator used for *early correlation*, so it can start collecting and pre-processing events immediately. Whenever a new device enters the system it performs the same steps as the other devices during initialization. After initializing our system, mobile devices will start transmitting events using 3G or GSM communication while at the same time searching for other devices in Wireless Lan range.

In our algorithm we take advantage of this and group the devices into local clusters, that collect events among each other. These local clusters consist of one Cluster-Head device and a set of Member devices. The Cluster-Head takes on the role of the relay and handles all 3G and GSM communication for the cluster devices. Member devices will use the cheaper Wireless Lan communication to transfer events to the Cluster-Head. There the events will be collected until a transmission to the server is scheduled, at which point the results of the *early correlation* will be sent to the server using 3G or GSM communication. Depending on the operator used, *early correlation* may happen parallel to the collection process or once before transmission.

In order to find these clusters we use a Wireless Lan broadcast message to reach all devices in range of the broadcasting device. Receivers of this message will answer the broadcaster with a message containing information about their status. Status information consists among others of information about whether the device is part of a cluster or not, and if so the cluster's size and the device's role in it.

The broadcasting device will collect answers for a short time to compensate for message delays while continuing to send events via 3G or GSM to the server. After that the device will count the number of replies that indicate devices in range that are not part of a cluster yet and check for messages from devices that occupy the role of Cluster-Head for already existing clusters. Depending on the systems parameters the broadcasting device will then either create a new cluster with other cluster-less devices or join an existing cluster by notifying a Cluster-Head device. If neither of both options is available, for example due to a lack of sufficient devices in the area, the device will continue to send events directly to the server using 3G or GSM communication for the time being and keep regularly looking for clusters in their proximity.

Our system incorporates mobile devices that are constantly moving. Devices will eventually move away from each other and clusters will dissipate. To compensate for this Cluster-Heads will have to perform some sort of maintenance operation to ensure Member devices will not just move out of communication range and continue sending events via Wireless Lan messages that are never received. We realize this by having a Cluster-Head regularly send a check-in message to its cluster's Member devices. This way Member devices will notice if they are out of reach, since they will no longer receive such messages.

If a Member realizes that it is no longer in range of its Cluster Head, it will switch back to being a cluster-less device and report events directly to the server using 3G or GSM again. Additionally it keeps searching regularly for other devices until it is able to join a cluster or build a new one.

Devices that are Members are able to safe energy due to cheaper Wireless Lan communication. Cluster-Heads however will actually use up more energy than they normally would due to additional computational cost for *early correlation* and additional maintenance communication. To better distribute those additional burdens over the whole system we will switch Cluster-Heads within a cluster from time to time. To do so the Cluster-Head will choose a Member device with sufficient power resources as new Head and inform all Cluster Members.

Since the Cluster-Head will have a new physical position, this may lead to some devices no longer being in range and some restructuring. However this will probably be necessary at some point anyway due to device mobility and this way the previous Cluster-Head will have an opportunity to compensate for the additional drain by taking on the role of a Member.

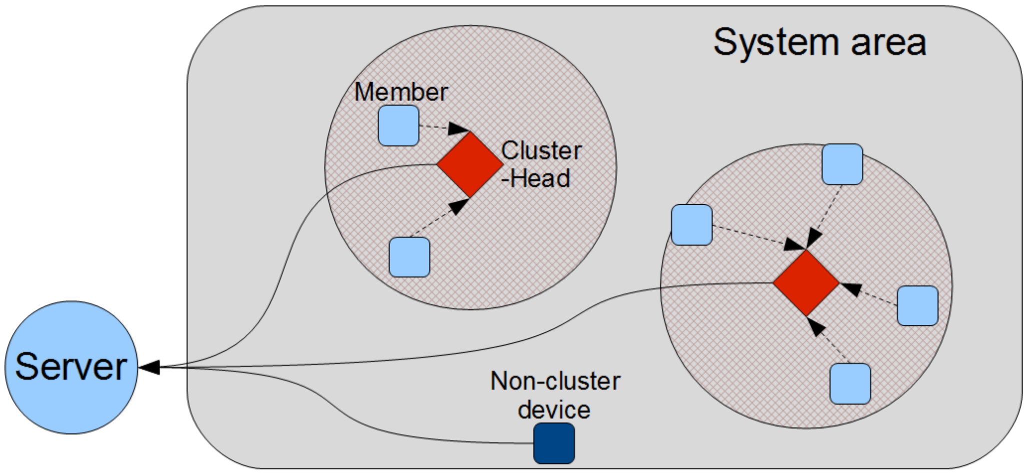


Figure 6.1: Local clusters in the CEP system.

Figure 6.1 shows a snapshot of a possible configuration of the CEP system using our algorithm. The grey area represents the physical space in which the devices are located. The red diamonds illustrate Cluster-Head devices that collect events from the cluster Members shown as light blue squares. The circles around the Head devices indicate their Wireless Lan communication range. The dark blue square represents a device that is currently not part of any cluster. Event transmission is depicted as arrows, where dotted lines are Wireless Lan and solid lines are 3G or GSM communication.

6.2 Initialization

During the initialization process, all mobile devices will make themselves known to the server. Depending on the nature of the system, the server will either subscribe to all participating mobile devices or the mobile devices will register with the server. In all cases the mobile devices will receive an address or identifier with which they can address the server using 3G or GSM communication. This makes sure that each device is able to transmit the events it will produce on its own without a relay.

To allow dynamic assignment of the Cluster-Head role, every mobile device will also receive all necessary operator information to fulfil the role of Cluster-Head. That information includes above all the operator that is used for *early correlation*. This is necessary since our system incorporates mobile devices, which means that we will not know which devices will be in Wireless Lan communication range of each other at different points in time,making it impossible to tell beforehand, which devices will take on the role of Cluster-Head.

Alternatively we could transfer the operator data on demand, whenever a device actually assumes the role of relay device for the first time. This might however delay the transmission of events to the server noticeably, depending on the amount of data the mobile device has to download from the server.

In case a device parts with the system it will unregister with the server, and may delete all operator data needed for *early correlation*. Alternatively our system offers the possibility to locally buffer the operator information, since it is more cost-effective than receiving them again from the server. In this case the operator information is stored on the mobile device for a certain amount of time and then deleted.

6.3 Cluster Finding

This section will describe the algorithm used to build or join clusters in detail. Mobile devices can either be a Cluster-Head, Cluster-Member or Non-Cluster devices, depending on their cluster membership and role. Cluster-Head and Cluster-Member devices are part of a cluster in which they hold the role corresponding to their name, while Non-Cluster devices are not part of a cluster at the moment.

Algorithms 6.2 and 6.3 are executed continuously on Cluster-Heads and cluster-less devices. Whenever a Non-Cluster device starts actively searching for a cluster, Algorithm 6.3 continues to run in the background. Algorithm 6.1 does not contain the operations performed by the process *HandleClusterRequestNonCluster* for purposes of abbreviation.

Algorithm 6.1 is executed regularly on mobile devices that are currently not part of a cluster.

During the search process we need to keep track of the Cluster-Head devices in range, whose clusters we may join. We also want to store all Ids of Non-Cluster devices in the proximity, in case we will be able to build a new cluster. For this purpose we initialize a set of variables at the beginning of the algorithm. *BestHeadCandidate* and *BestHeadClusterSize* are used to keep Track of possible Cluster-Head devices, while *PossibleMemberList* will store the Ids of Non-Cluster devices in range. Since being a Cluster-Head puts more strain on a devices energy resources than normal operation we want to restrict the number of devices that are able to become Head of a new cluster to those with a certain amount of energy left. *MinHeadPowerLevel* is a system Parameter that defines how much energy a device has to have left in order to do that. The ability to create new clusters is expressed by *NewCluster*.

To find other devices in range, the requesting device will broadcast a *ClusterRequest* message containing the searching devices own unique Id *OwnId* and *NewCluster*. To allow other devices to respond before deciding on how to proceed the device will collect answers for a short time. In Algorithm 6.1 this is done by processing received messages while the Timer is running. Duration of the Timer is defined by the system.

The *ClusterRequest* message is received by mobile devices in Wireless Lan communication range. These devices are either Head of a cluster, Member of a cluster or not part of any cluster. Depending on their role, they will execute one of the Algorithms 6.2 or 6.3. Cluster-Member devices will simply discard *ClusterRequest* messages, since their cluster's Head device manages everything.

Cluster-Heads executing Algortihm 6.2 will, upon receiving a *ClusterRequest* message check their own cluster's size against the system parameter *MaxClusterSize*. *MaxClusterSize* restricts the maximal cluster size allowed in the system in order to keep them at a manageable size and prevent bottlenecking. If the cluster is not yet full, the Head device invites the requesting device to join by sending it a *JoinOffer* message containing the Head device's Id *OwnId* and .

When Non-Cluster devices executing Algorithm 6.3 receive a *ClusterRequest* message, they will first check the parameter *NewCluster*. If *NewCluster* is *True* that means the devices could possibly create a new cluster together since the requesting device has enough power to become a Head-Device. In this case the Non-Cluster device will answer with a *MemberRequest* message containing its own Id *OwnId*, indicating its willingness to form a new cluster. Additionally the Non-Cluster device will store the requesting device's Id in *NewHeadCandidates*, to keep track of all devices it offered to build a new cluster with. In case *NewCluster* is *False* the *ClusterRequest* message is simply discarded.

Algorithmus 6.1 Cluster Request from Non-Cluster device

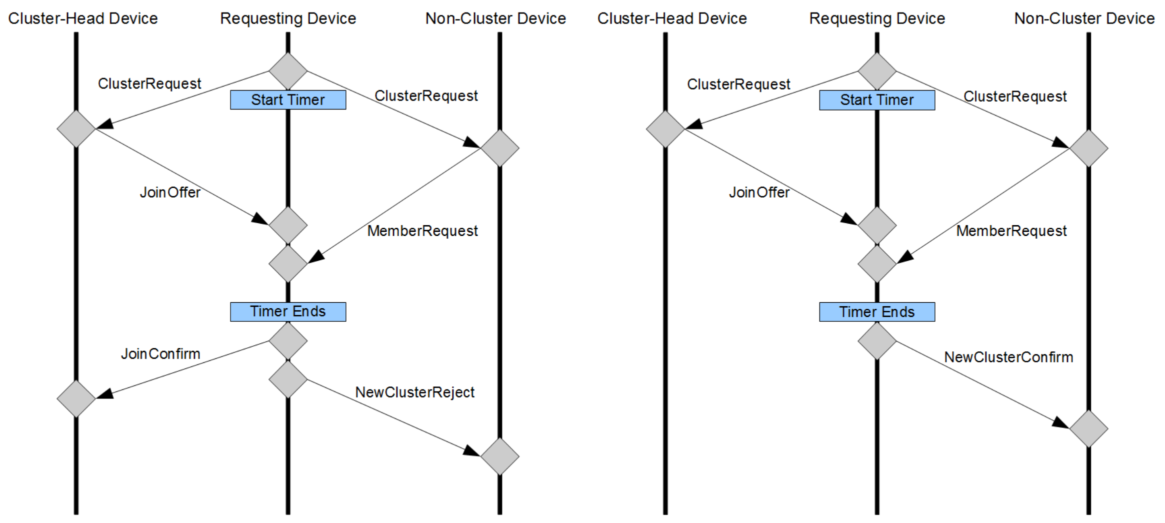
```
procedure ClusterRequest
    BestHeadCandidate ← $NULL$
    BestHeadClusterSize = MaxClustersize
    PossibleMemberList ← $NULL$
    if (DevicePowerLevel > MinHeadPowerLevel) then
        NewCluster ← $True$
    else
        NewCluster ← $False$
    end if
    Broadcast ClusterRequest($OwnId, NewCluster$)
    Start Timer
    while ($Timer\ is\ running$) do
        if (Receive JoinOffer($HeadId, ClusterSize$)) then
            if ($Clustersize$ < BestHeadClusterSize) then
                BestHeadCandidate ← $HeadId$
                BestHeadClusterSize = $ClusterSize$
            end if
        else if (Receive MemberRequest($DeviceId$)) then
            PossibleMemberList.add($DeviceId$)
        else if (Receive Part($DeviceId$)) then
            PossibleMemberList.remove($DeviceId$)
        end if
    end while
    if (PossibleMemberList.size >= MinClusterSize) then
        for all ($Id \in$ PossibleMemberList) do
            Send NewClusterConfirm($OwnId$) → $Id$
        end for
        DeviceRole = $Head$
        ClusterMember ← PossibleMemberList
    else
        if (BestHeadCandidate ≠ $NULL$) then
            Send JoinConfirm($OwnId$) → BestHeadCandidate
            Set Cluster-Head(BestHeadCandidate)
            DeviceRole = $Member$
        else
            Reschedule ClusterRequest(SearchInterval)
        end if
        for all ($Id \in$ PossibleMemberList) do
            Send NewClusterReject($OwnId$) → $Id$
        end for
    end if
end procedure
```

While waiting for the Timer to stop, the requesting device will collect the messages sent in return by the other devices.

Whenever a *JoinOffer* message is received, the device will check its *Clustersize* parameter against *BestHeadClusterSize*, which contains the size of the currently selected cluster, the requesting device might join. If the sender's cluster size is smaller than that of the currently selected, it is preferred as new cluster. This way we divide devices more equally among existing clusters. As a result *BestHeadCandidate* will be assigned the sender's Id *DeviceId* and *BestheadClusterSize* is updated with *Clustersize*.

In case a *MemberRequest* is received, the *DeviceId* contained in the message will simply be added to *PossibleMemberList* to keep track of devices that are available to build a new cluster. A *Part* message however indicates that a device that was previously available has now already been used in construction of a different cluster. Hence *DeviceIds* contained in *Part* messages are removed from *PossibleMemberList*

Once the Timer has stopped, the requesting Device will continue to execute Algorithm 6.1. From this point on there will be 3 different courses of action for the requesting device, depending on the received answers.



(a) Flowchart in case of joining existing Cluster **(b)** Flowchart in case of building new Cluster

Figure 6.2: Flowcharts for Cluster finding process

The first one is illustrated by Figure 6.2b and will be taken in case enough Non-Cluster devices responded to build a new cluster. This is determined by checking the number of entries in *PossibleMemberRequest* against *MinClusterSize*, which is a system parameter that indicates the minimal cluster size that is allowed in our system. If enough devices are available it proceeds to send a *NewClusterConfirm* message containing its own Id *OwnId* to all participating Non-Cluster device Ids stored in *PossibleMemberRequest*. Finally it assumes its new role as Cluster-Head by setting its *DeviceRole* indicator to *Head* and converting *PossibleMemberList* to *ClusterMember*. Since Cluster-Head devices only invited the requesting device to join but did not actually add it to their Members, no extra message is necessary to inform them of the device not joining.

Non-Cluster devices that receive a *NewClusterConfirm* message will send a *Part* message to any other device they told they were available, informing them this is no longer the case. They then proceed to assume their new role as Cluster-Member by setting the requesting device as their new Cluster-Head and their own role to *Member*.

Algorithmus 6.2 Handle Cluster Request on Cluster-Head device

```
procedure HANDLECLUSTERREQUESTHEAD
    while DeviceRole = $Head$ do
        if (RECEIVE CLUSTERREQUEST($DeviceID, NewCluster$)) then
            if (OwnClusterSize < MaxClusterSize) then
                SEND JOINOFFER($OwnId, OwnClusterSize$) $\rightarrow$ $DeviceId$
            end if
        else if (RECEIVE JOINCONFIRM($DeviceId$)) then
            ClusterMember.ADD($DeviceId$)
        end if
    end while
end procedure
```

Algorithmus 6.3 Handle Cluster Request on Non-Cluster device

```
procedure HANDLECLUSTERREQUESTNONCLUSTER
    NewHeadCandidates $\leftarrow$ $NULL$
    while DeviceRole = $None$ do
        if (RECEIVE CLUSTERREQUEST($DeviceID, NewCluster$)) then
            if ($NewHead$) then
                SEND MEMBERREQUEST($OwnId$) $\rightarrow$ $DeviceId$
                NewHeadCandidates.ADD($DeviceId$)
            end if
        else if (RECEIVE NEWCLUSTERCONFIRM($DeviceId$)) then
            NewHeadCandidates.REMOVE($DeviceId$)
            for all ($Id \in$ NewHeadCandidates) do
                SEND PART($OwnId$) $\rightarrow$ $Id$
            end for
            SET CLUSTER-HEAD($DeviceId$)
            DeviceRole = $Member$
        else if (RECEIVE NEWCLUSTERREJECT($DeviceId$)) then
            NewHeadCandidates.REMOVE($DeviceId$)
        end if
    end while
end procedure
```

Figure 6.2a illustrated the second possible outcome. In this case the number of available devices to form a new cluster is insufficient. The requesting device then continues to check if *BestHeadCandidate* is assigned an Id of a Cluster-Head. If so the requesting device has received at least one *JoinOffer* message during the timer phase and is able to join an already existing cluster. It does so by sending a *JoinConfirm* message containing its *OwnId* to the Id stored in *BestHeadCandidate*. To assume its role as a Cluster-Member it then sets its Cluster-Head to *BestHeadCandidate* and its own role to *Member*. Since no new cluster has been built the requesting device also informs all Non-Cluster devices stored in *PossibleMemberList* about that fact, by sending a *NewClusterReject* message containing its *OwnId*. Non-Cluster devices receiving such a message will remove the message's *DeviceId* from their *NewHeadCandidates*.

The last possible outcome for the requesting device is that neither enough Non-Cluster devices are in range to build a new cluster nor did it receive a *JoinOffer* message. As a result the device has no possibility to join a cluster at this point it will reschedule the clusterfinding process at a later time, which is defined by the system parameter *SearchInterval* and inform all devices in its *PossibleMemberList* that no cluster will be created by sending them a *NewClusterReject* message. Without any cluster association the device will continue to send events directly to the server using 3G or GSM communication.

Concurrency issues will not occur in our system, since mobile devices will only accept messages they expect in their current role. As an example we look at the case that two clusters declare themselves the new Cluster-Head of a cluster at the same time, with each cluster containing the other device. Before sending the *NewClusterConfirm* message to the other mobile devices involved in the construction, each device will become Cluster-Head. As Cluster-Head, Algorithm 6.3 will no longer be executed on the device and any *NewClusterConfirm* messages will not be processed. and both devices will continue to act as Cluster-Head. The fact that each device is not a member of the other device's cluster will be detected by the maintenance algorithms introduced in the next section.

Parameters

Describing the algorithm we mentioned a number of system variables. Namely these are *MinHeadPowerLevel*, *MaxClusterSize*, MinClusterSize and *SearchInterval*.

MinHeadPowerLevel is supposed to restrict the number of devices that become Cluster-Heads to those with enough power. Without this parameter every device would try to build a new cluster, including those that are already very low on power. Would such a device become a Cluster-Head it could quickly run out of energy completely, since it would have to perform *early correlation* and cluster maintenance operations as well as the normal event transmission. By selecting devices with sufficient energy resources as Heads we assure that all devices will be able to report event for as long as possible. Choosing a high value for *MinHeadPowerLevel* could however result in a lack of possible Cluster-Heads. Devices would regularly search for clusters but would rarely be able to build new ones. Over time this might lead to a sparseness of clusters for devices to join. If the probability of finding a cluster is too low, the energy used to find clusters will be mostly wasted.

Searchinterval defines the frequency with which devices search for clusters. If *SearchInterval* is set so that devices are searching for clusters very frequently they will find a cluster as soon as possible, but will probably waste a decent amount of energy doing so. A new search is only feasible if the device positions in the system have changed to a certain amount, otherwise the device would only be answered by the same set of devices over and over. Change of positions in the system is the result of device movement, and as such *Searchinterval* should be linked to the overall mobility of the system. If devices are mostly moving slowly it is sensible to perform searches less frequently since positions of devices and cluster coverage will also change slowly. Due to this results of frequent searches are likely to be very similar and not produce any new results. The faster devices move the faster will positions change and new local accumulations of devices be formed. The more frequent change will also warrant more frequent searches.

With faster moving devices, cluster also tend to disperse faster. Which will cause more expenses in terms of cluster maintenance which we will discuss in detail in the next section. In general if cluster disperse faster we need to check more often if all Members are still in range. If they are not they will have to search for a new cluster again, which might be dispersed again soon. This search/maintenance cycle might cause a decent amount of communication overhead if *Searchinterval* is chosen to small.

Another criteria for choosing *Searchinterval* is the frequency with which events are produced. Consider a system of fast moving devices that very frequently have to search for clusters, which in turn will disperse quickly. Now if this system only produces events with a very low frequency the systems maintenance cost is wasted for the most part since devices will join and leave cluster without ever sending even one event. In this case a solution with a low *Searchinterval* that will only search for cluster right before events are sent may be better suited.

MinClusterSize and MaxClusterSize are both system parameters that control the size of clusters. *MinClusterSize* controls as the name implies the minimum number of devices a cluster has to consist of. To better understand the purpose of the parameter we recall that every Cluster-Head also has to perform early-correlation of some sort, additionally to handling all communication. This *early correlation* will cost power since it needs to use the devices processor. We want to safe energy overall by having Members use Wireless Lan, which however might not be enough to account for the additional drain if only one Member is actually doing so. In order to assure that power is saved overall when building a cluster we need to estimate the additional energy drain caused by *early correlation* and the savings by using Wireless Lan. From these two factors we can then derive a sensible number of minimal devices a cluster has to possess in order to be profitable. If we assume for example that *early correlation* is performed once when every Cluster-Member transmitted an event, we can estimate the clusters savings by applying the formula: $(Number\ of\ Members) * (Savings\ per\ WLAN\ Message) - (Early\ Correlation\ Cost)$.

The bigger the cluster the more energy we can save by using Wireless Lan. For this reason we want to keep *Maxclustersize* as large as possible. However we are using mobile devices like smart-phones in our system, whose computational power is still limited. A Cluster-Head will have to perform early-correlation and handle a number of incoming messages due to incoming events and cluster maintenance. Depending on the form of early-correlation done by the Head device and amount of communication a device might get overwhelmed, creating a bottleneck in the system since all of the cluster communication has to pass through it. In order to keep this from happening we try to restrict cluster size to a reasonable number. We can derive a value for *MaxClusterSize* by estimating the computational effort of early-correlation and looking at the devices hardware specifications.

6.4 Cluster Maintenance

The previous chapter explained how to find and build device clusters. In this chapter we will explain the algorithms used for cluster maintenance. The first and most important task of cluster maintenance is to keep track if Member devices are still in Wireless Lan communication range of the Cluster-Head. Without doing so devices might drift apart and continuously send events over Wireless Lan that will never be received without noticing it. The second part of the maintenance is the cycling of Cluster-Heads in order to distribute the additional workload caused by *early correlation* and relaying all events more evenly among the system.

Our goal is to safe as much energy as possible in the overall system. Maintenance can cause a lot of communication, which causes additional energy drain. To keep communication to a minimum we will introduce two classes of maintenance algorithms. The first class consists of a number of lazy algorithms that keep communication to a minimum but may result in lost events and is described in Subsection 6.4.1.The second class is a set of eager algorithms described in Subsection 6.4.2, that guarantee maintenance without loosing messages, but will produce more network traffic.

6.4.1 Lazy Maintenance

In this section we will describe the algorithms used to implement lazy cluster maintenance. We call these algorithms lazy because we will not make sure that every event reaches its goal, but instead try to keep the number of sent messages to a minimum in order to safe the maximum amount of energy in the system.

Subsubsection 6.4.1 describes the realization of range checks, while Subsubsection 6.4.1 introduces the algorithms used to restructure clusters to better distribute the additional power consumption of Cluster-Heads.

Rangecheck

As soon as a Device takes on the role of Cluster-Head it begins to execute Algorithm 6.4.

To let Cluster-Members know if their Cluster-Head is still in range, Cluster-Heads regularly send a *CheckInMessage* to all members. They do so every time a timer called

SendRangeCheckTimer expires. The duration of this timer and as such the frequency with which we send these messages is determined by the system parameter *SendRangecheckInterval*. The Cluster-Head will check if its members are still in range by using another timer called *DeviceTimeoutTimer*. One of these timers is running on the Cluster-Head for each of its members.

When a Non-Cluster device joins a cluster and becomes a Cluster-Member, it starts executing Algorithm 6.5.

To check if their Cluster-Head is still in range they wait for *CheckInMessages* sent by their Cluster-Head. If they do not receive such a message regularly, they consider the Cluster-Head to be out of range. The amount of time that may pass between receiving these messages is given by a timer called *ReceiveRangecheckTimer*, whose duration is determines by the system parameter *ReceiveRangeCheckInterval*. Since *CheckInMessages* are sent in an interval determined by *SendRangeCheckInterval*, *ReceiveRangeCheckInterval* should be set a little larger than that to compensate message delays.

After all the timers mentioned above are initialized, the devices will commence with the actual maintenance operations until, in case of the Head device, it either stops being a Cluster-Head or a certain amount of power, defined by the system parameter *HeadCyclePowerAmount*, has been spent. By limiting the duration of being a Cluster-Head, using the parameter *HeadCyclePowerAmount* we distribute the extra workload more evenly among all devices. Member devices will keep performing maintenance operations until they no longer occupy the role of a Cluster-Member.

While performing maintenance three situations may occur on the Cluster-Head device, that need to be resolved.

The first event we need to resolve is the expiration of the *SendRangeCheckTimer*. In this case the Cluster-Head will send a *CheckInMessage* to all cluster members. Afterwards it will restart the *SendRangeCheckTimer* to schedule the next check-in.

Regular *CheckInMessages* let Cluster-Member devices know if they are still in range of the Cluster-Head or not. Upon receiving a *CheckInMessage*, a member will simply restart its own *ReceiveRangeCheckTimer* and wait for the next check-in. Should no *CheckInMessage* be received in time and *ReceiveRangeCheckTimer* expire, we can consider the head device to be out of range. In this case the member device will abandon its role and become a Non-Cluster device, by setting its own *DeviceRole* to *None* and scheduling a *ClusterRequest*.

Algorithmus 6.4 Lazy Range Check on Cluster-Head

```
procedure LazyRangeCheckHead
    Start SendRangeCheckTimer($(SendRangeCheckInterval)$)
    for all $(Id \in ClusterMember)$ do
        Start DeviceTimeoutTimer($(Id, TimeoutInterval)$)
    end for
    while (DeviceRole = $Head$) do
        if (Finished SendRangeCheckTimer) then
            for all $(Id \in ClusterMember)$ do
                Send CheckInMessage($OwnId$) $\rightarrow Id$
            end for
            Restart SendRangeCheckTimer($(SendRangeCheckInterval)$)
        else if (Receive EventData($DeviceId$)) then
            Reset DeviceTimeoutTimer($(DeviceId)$)
        else if (Finished DeviceTimeoutTimer($(DeviceId)$)) then
            ClusterMember.remove($DeviceId$)
            if (ClusterMember.size() $<$ MinClustersize) then
                for all $(Id \in ClusterMember)$ do
                    Send Abandon($OwnId$) $\rightarrow Id$
                end for
                DeviceRole = $None$
                Schedule ClusterRequest($SearchInterval$)
            end if
        else if $(HeadCylePowerAmount\ spent)$ then
            RestructureHead
        end if
    end while
end procedure
```

The second event on the Cluster-Head is the reception of *Event-Data* from any Cluster device. As long as the head device regularly receives *Event-Data* from a cluster member, that member is still in Wireless Lan range, and we only need to restart the *DeviceTimeoutTimer* for the respective device. Should a *DeviceTimeoutTimer* expire for a specific member device, that event has not send any data for an unusual amount of time. In this case we consider the device to be out of range and remove it from *ClusterMember*.

Since our cluster just lost a member we also have to check if the cluster still is big enough to satisfy the system parameter *MinClusterSize*. If so no further action has to be taken. However if the cluster just became to small we decide to dissolve it. We do so by sending an *Abandon* message to all remaining cluster members, converting the Cluster-Head to a Non-Cluster device and scheduling a *ClusterRequest*.

Members that receive an *Abandon* message will also become Non-Cluster devices and schedule *ClusterRequests*.

Algorithmus 6.5 Lazy Range Check on Cluster-Member

```
procedure LazyMaintenanceMember
    Start ReceiveRangeCheckTimer($ReceiveRangeCheckInterval$)
    while (DeviceRole = $Member$) do
        if (Receive CheckInMessage($DeviceId$)) then
            Reset ReceiveRangeCheckTimer
        else if (Finished ReceiveRangeCheckTimer) or (Receive Abandon($DeviceId$)) then
            DeviceRole = $None$
            Schedule ClusterRequest(SearchInterval)
        else if (Receive InformNewHead($DeviceId, DeviceList$)) then
            LazyRestructureMember($DeviceId, DeviceList$)
        end if
    end while
end procedure
```

Restructure

Once a Cluster-Head spent the amount of power defined by the system parameter *Head-CylePowerAmount* it will relinquish its role as head by calling the procedure *RestructureHead*, which is described in Algorithm 6.6.

RestructureHead starts by assigning the first cluster member to the variable *NewHead*. *NewHead* will contain the Id of the device that will replace the current head. Afterwards it checks all entries of *ClusterMember* for devices whose power resources are higher than those of the device currently stores in *Newhead*. If such a device is found *NewHead* is instead assigned the device Id with more power.

For the Cluster-Head to be able to monitor the members power levels, Cluster-Members will transmit their current power level to the server every time they send *EventData*. This will not cause significant energy costs since appending one number to the message will not change transmission costs noticeably.

After selecting its successor the current Cluster-Head will become a member device itself and adds its *OwnId* to the list of cluster members *ClusterMember*. By setting *NewHead* as its Cluster-Head the device finishes its role conversion. To inform its former cluster members about the change it goes on to send them an *InformNewHead* message containing the new head's Id *NewHead* and the Ids of all members *ClusterMember*.

The member devices will receive the *InformNewHead* message while executing their maintenance Algorithm 6.5, and when doing so will start the procedure *LazyRestrucuture-Member* described in Algorithm 6.7.

LazyRestrucutureMember will first check if the receiving devices *OwnId* matches the new heads Id *NewHead*. If so that means that the receiving device was chosen as the new head. In this case the device will assume its new role as Cluster-Head and assign the List of members contained in the message as *DeviceList* to its own list of members *ClusterMember*. Additionally the device will have to remove its *OwnId* from the member list, since the former head did not already do so. Finally the new head will immediately send *CheckInMessages* to all members to refresh their *ReceiveRangeCheckTimers*. Devices whose id do not match *NewHead* will simply have to set *NewHead* as their new Cluster-Head and reset their *ReceiveRangeCheckTimer* because the new Head will also start a new *SendRangeCheckTimer* as it assumes its new role.

Algorithmus 6.6 Lazy Restructure on Head

```
procedure RestructureHead
    NewHead ← ClusterMeber.first
    for all ($Id \in ClusterMember$) do
        if ($Id$.PowerLevel> $NewHead$.powerlevel) then
            NewHead ← $Id$
        end if
    end for
    DeviceRole = $Member$
    Set Cluster-Head($NewHead$)
    ClusterMember.add($OwnId$)
    for all ($Id \in ClusterMember$) do
        if $Id \neq OwnId$ then
            Send InformNewHead($NewHead, ClusterMember$) → $Id$
        end if
    end for
end procedure
```

Algorithmus 6.7 Lazy Restructure on Member

```
procedure LazyRestructureMember($DeviceId, DeviceList$)
    if ($OwnId = NewHead$) then
        DeviceRole = $Head$
        ClusterMember ← $DeviceList$
        ClusterMember.remove($OwnId$)
        for all ($Id \in ClusterMember$) do
            Send CheckInMessage($OwnId$) → $Id$
        end for
    else
        Set Cluster-Head($DeviceId$)
    end if
end procedure
```

Parameters

Implementing the maintenance procedures we also introduced a new set of system parameters we will now discuss in detail.

SendRangeCheckInterval and ReceiveRangeCheckInterval are directly related to each other. The former determines how frequently a Cluster-Head sends *CheckInMessages* to its members, while the later sets how long those members wait for these messages. Therefore *ReceiveRangeCheckInterval* should always be at least as big as *SendRangeCheckInterval*. Considering message delays we actually want to make sure that *ReceiveRangeCheckInterval* is bigger than its counterpart. How much bigger will depend on expected delays and how sure we want to be that delayed messages will be received in time. In general we do not want the difference to be too big, or member devices will take to long to realize that they are out of range.

Deciding on how to choose *SendRangeCheckInterval* presents similar problems as choosing the system parameter *Searchinterval*. Frequent check-ins will result in earlier detection of members that are out of range, but will cause more communication and consume more energy. Rare check-ins will possibly result in devices being out of range for some time before noticing it but will produce less messages and safe more energy. Also any events that were sent to the Cluster-Head by the members during this time will be lost using lazy maintenance. To find a sensible value for *SendRangeCheckInterval* we can look at the systems mobility and how crucial loosing some events actually is. In systems with high mobility devices tend to move out of range faster due to the high speed of the member devices, so frequent check-ins will noticeably reduce the number of lost events. In systems with slow devices changes are less probable to happen frequently and we can choose a larger *SendRangeCheckInterval*.

TimeoutInterval determines how long a head device will wait to receive events from devices before considering them out of range. This parameter is directly linked to the frequency with which events are produced by the system. Its is not reasonable to set a *TimeoutInterval* smaller than the frequency we expect the devices to produce events or most device would be considered out of range immediately. So we want to keep *TimeoutInterval* as least as big as the expected event frequency. Depending on expected message delay and accuracy of the estimation for event frequency it is sensible to increase the parameter by an appropriate amount to compensate for these factors. In general we rather want to choose *TimeoutInterval* a little too big than too small, since head devices will not expect events messages from devices, that are not in their member list. They should however be able to perform *early correlation* even if one of their members did not send any *EventData* messages.

HeadCyclePowerAmount indirectly defines how frequently the role of Cluster-Head is switched in a cluster. By choosing higher values we keep the same head device for a longer time which increases the systems stability. After each restructuring a number of member devices will probably end up out of range of the new Cluster-Head, since the new head device will be at a different position. In this lazy implementation we do not explicitly check which members are still in range after restructuring, so events may be lost until the remaining duration of the, now out of range, member devices *ReceiveRangeCheckInterval* has passed. Higher values for *HeadCyclePowerAmount* will lead to certain devices' energy resources being drained noticeably faster than others, since they will possibly stay head for a long time. Smaller values will lead to a better overall distribution of energy consumption but due to the communication cost of restructuring will also drain more energy from the system. Head changes also seem more sensible in systems with low mobility, since cluster are more probable to persist over a longer period. In system with high mobility and fast moving devices clusters are more probable to disperse after a short amount of time, in which case a new head might be selected upon building a new cluster anyway.

6.4.2 Eager Maintenance

While the lazy maintenance algorithms minimize communication they may loose events at certain points. This is unacceptable if the CEP system must not loose any events. To achieve lossless event transmission we will now introduce a set of alternative eager algorithms that can be used in place of their lazy counterparts. However guaranteed event transmission comes at the prize of increased maintenance communication and as such will cost more energy.

Rangecheck

The basic idea behind this new set of algorithms is the replacement of certain timer based control bursts by using direct confirmation messages. Algorithm 6.8, which replaces Algorithm 6.4, for example does not start by initializing a SendRangeCheckTimer, since there will not be regular *CheckinMessages* to the cluster members. Instead procedure *EagerRangeCheckHead* will directly respond to every received *EventData* by sending an *EventReceived* message as a direct answer to the member device it received the event from.

Timeout of members will be handled the same way as in Algorithm 6.4. A timer *DeviceTimeoutTimer* with the duration *TimeoutInterval* will be started for each member device. If an event is received while the timer is still running, the timer will simply be reset. Is this not the case and the timer expires, the head considers the member device to be out of range and removes it from its member list. Since the cluster just became smaller the head device will check if the cluster is still big enough to mainatin by comparing its size to the system parameter *MinClusterSize*. If the cluster is too small it will be disbanded by sending an *Abandon* message to the members. Finally the head device will become a cluster-less device and schedule a *ClusterRequest*.

Member devices will have their own means of detecting whether they are still in range of their Cluster-Head and if not will send events directly to the server, so detecting a member that is no longer in range on the head device will not cause event loss. Keeping track of members in range on the head is only used to determine the point at which the cluster has become too small to maintain.

Just as in the lazy variant we distribute the additional workload of a head device by having it hand over its role to a member device, after it used up a certain amount of power. The difference is that we call Algorithm 6.6 instead of its lazy counterpart.

Algorithmus 6.8 Eager Range Check on Cluster-Head

```
procedure EagerRangeCheckHead
    for all $(Id \in ClusterMember)$ do
        Start DeviceTimeoutTimer$(Id, TimeoutInterval)$
    end for
    while (DeviceRole $=$ $Head$) do
        if (Receive EventData($DeviceId$)) then
            Send EventReceived($OwnId$) $\rightarrow$ $DeviceId$
            Reset DeviceTimeoutTimer$(DeviceId)$
        else if (Finished DeviceTimeoutTimer($DeviceId$)) then
            ClusterMember.remove($DeviceId$)
            if (ClusterMember.size() $<$ MinClustersize) then
                for all $(Id \in ClusterMember)$ do
                    Send Abandon($OwnId$) $\rightarrow$ $Id$
                end for
                DeviceRole $=$ $None$
                Schedule ClusterRequest($SearchInterval$)
            end if
        else if $(HeadCylePowerAmount\ spent)$ then
            RestructureHead
        end if
    end while
end procedure
```

CheckInMessages no longer exist, as a result Algorithm 6.9, which replaces Algorithm 6.5, no longer needs a *ReceiveRangeCheckTimer*. Instead we initialize the variable *Received* with *False*, which we will use to check if we received a confirmation message from the head device after sending an event.

We want to assure that every event will be delivered to the server. In order to do so we start a timer *Timeout* every time we send an event to the Cluster-Head. If we receive an *EventReceived* message from the head device while *Timeout* is still running we know the event will be relayed and set *Received* to *True* to indicate this fact. In this case we will simply reset *Received* to *False* after the timer has finished. Should on the other hand no confirmation message arrive and *Received* still be *False* after the timer expires, we assume that the head is no longer in range and never received the event. Hence the member device will leave the cluster and become a cluster-less device and schedule a new *ClusterRequest*. To guarantee the server will receive the event it will also resend the event directly to the server using 3G or GSM communication.

The rest of Algorithm 6.9 remains fairly unchanged. In case the device receives an *Abandon* message the device will abandon its member role and schedule a *ClusterRequest*. The only other difference is that in contrast to its lazy equivalent Algorithm 6.9 will call Algorithm 6.10 upon receiving an *InformNewHead* message.

Algorithmus 6.9 Eager Range Check on Cluster-Member

```
procedure EAGERMAINTENACEMEMBER
    Received = $False$
    while DeviceRole = $Member$ do
        if ($just\ sent\ EventData$) then
            START TIMEOUT
            while ($Timeout\ is\ running$) do
                if (RECEIVE EVENTRECEIVED($DeviceId$)) then
                    Received = $True$
                end if
            end while
            if not (Received) then
                DeviceRole = $None$
                SCHEDULE CLUSTERREQUEST(SearchInterval)
                RESEND EVENTDATA(OwnId)$\rightarrow ServerId$
            end if
            Received = $False$
        else if (RECEIVE ABANDON($DeviceId$)) then
            DeviceRole = $None$
            SCHEDULE CLUSTERREQUEST(SearchInterval)
        else if (RECEIVE INFORMNEWHEAD($DeviceId, DeviceList$)) then
            EAGERRESTRUCTUREMEMBER($DeviceId, DeviceList$)
        end if
    end while
end procedure
```

Restructure

Algorithms 6.6 and 6.10 handle the restructuring of a cluster when a head device spent enough energy performing its task.

Since no timers are used in Algorithm 6.6 no modifications are necessary. We can simply use the same algorithm as in the lazy approach. First the current Cluster-Head chooses the member with the highest remaining amount of energy. Then it quits being a Cluster-Head and declares itself a Cluster-Member, before adding its *OwnId* to *ClusterMember*. Finally it sends an *InformNewHead* message to all previous Cluster-Members, to inform them that a new Cluster-Head has been appointed.

The Cluster-Members will receive the *InformNewHead* message and react by executing Algorithm 6.10.

Algorithm 6.10 differs from Algorithm 6.7 only in the point that the newly appointed Cluster-Head does not have to send a *CheckInMessage* to the other members, because it no longer needs to refresh any *ReceiveRangeCheckTimer*.

If the receiving members *OwnId* matches the *DeviceId*, which was contained in the *InformNewHead* message, the device will declare itself Cluster-Head. Then it will update its member list *ClusterMember* with the *DeviceList*, which was also contained in the message. Finally, since the previous head did not already do so, the new Cluster-Head will have to remove its *OwnId* from the list of members.

If the parameter *DeviceId* does not match the members *OwnId*, it will remain a Cluster-Member and simply update its Cluster-Head.

If a Cluster-Member is out of range of the new Cluster-Head, this will be detected by the time the member tries to send the next *EventData* message. If the device really is out of range it will not receive a *EventReceived* message in time and become a Non-Cluster device before resending the event directly to the server.

Algorithmus 6.10 Eager Restructure on Member

```
procedure EagerRestructureMember($DeviceId, DeviceList$)
    if ($OwnId = NewHead$) then
        DeviceRole = $Head$
        ClusterMember $\leftarrow$ $DeviceList$
        ClusterMember.remove($OwnId$)
    else
        Set Cluster-Head($DeviceId$)
    end if
end procedure
```

7 Evaluation

In this chapter we will evaluate the implementation of *relayed event transmission* with *early correlation* by comparing simulation results. We will examine how the different system parameters we introduced will effect the performance of the system and under which circumstances *relayed event transmission* with *early correlation* can be applied profitably. Section 7.1 will introduce the setup and software used for our simulations, while 7.2 present the results of a number of simulations using differing values for our systems parameters.

7.1 Simulation Setup

To evaluate our system, we conducted simulations using the OMNeT++ [Var01] simulation system, including the *Inetmanet* project, which allows the simulation of networks consisting of mobile devices. For our simulations we observed a restricted spacial area. In this area we placed a fixed number ob mobile devices, executing the *Lazy* variants of the algorithms presented in Chapter 6.

Devices produce and send events at a fixed frequency, which is given by the simulation parameter *EventInterval*. In order to transmit these events each device is able to send Wireless Lan messages over a range of 100 meters. 3G and GSM transmission range is unlimited. During the simulation messages are delivered without delay and no messages are lost due to transmission errors. Since we have no delay to compensate we will simply work with the parameter *RangeCheckInterval*, which will replace *SendRangeCheckInterval* ans *ReceiveRangeCheckInterval* during this chapter.

We simulate mobility of the system using the built in *MassMobility* model, which is conform with the model used in [PW99]. Devices will start moving in an initial direction with the speed defined by the simulation parameter *Speed*. Afterwards each device will change its direction every five seconds. This change is not completely random, instead to better simulate natural movement the current course of a device will be altered by at most 30 degrees. Whenever a device hits the borders of our simulation area it is reflected.

To measure energy in the system, devices are initialized possessing a fixed amount of energy. Since we are only interested in the performance of our algorithms we do not consider energy consumption of other device operations. To estimate the power a device has consumed during the simulation, we keep track of the number and type of sent and received messages for each device and apply the energy model we introduced in Chapter 3. Since we cannot reliably assume how much power operators used for *early correlation* will consume, we will concentrate on measuring the energy consumption of the communication in the system.

We also assume that messages sent during the simulation have a consistent size of 50Kb, otherwise we would not be able to apply our energy model correctly. This means that our systems implements operators of either the *Filter* or *Modifiable* class, that keep messages to the server small.

If not specified otherwise each simulation performed in this chapter covered a timespan of 1000 seconds. Mobile devices will be initialized with a power capacity of 10000 Joules and are moving with a speed of 5 meters per second.

The remaining system parameters required for our algorithm are by default set to the following values : *MinHeadPowerLever* is set to 4000 Joule when simulating performance for 3G transmissions and to 3000 Joule when simulating GSM. *SearchInterval* and *RangecheckInterval* are typically both set to 10 seconds. *MinClusterSize* is usually set to 3 while *MaxClustersize* per default 20. Finally *HeadCyclePowerAmount* is assigned a value of 500 Joule.

Simulations are performed 5 times with differing random number generator seeds. All results presented in this chapter are averaged over those 5 simulation runs.

7.2 Results

7.2.1 Performance for high-frequency event transmission

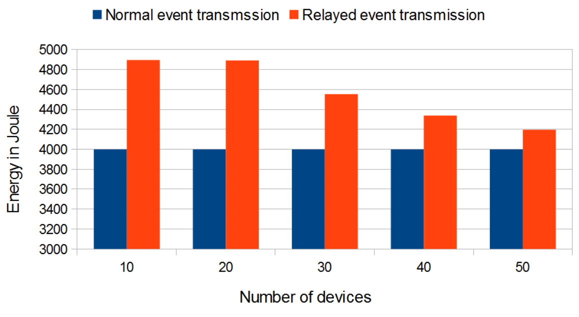
In this subsection we compare performance of our algorithm in case of high-frequency event transmission for a differing number of mobile devices. We also vary the size of the simulation area to better evaluate the performance of our algorithm. Events are produced and transmitted every 1 second, which means that every device will send 1000 events.

We simulated spatial areas of 250 square meters, 500 square meters and 750 square meters. Each time the area was sequentially populated by 10, 20, 30, 40 and finally 50 mobile devices. Each combination of area size and number of devices was simulated using the energy model for high-frequency 3G transmissions and using the model for high-frequency GSM.

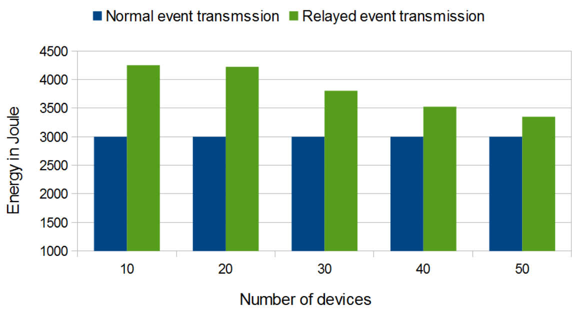
Figure 7.1 illustrates the results of these simulations. For the smallest area of 250 square meters, mobile devices using *relayed event transmission* had on average more energy left than when using the normal transmission model. This can be seen in Figures 7.1a and 7.1b, which shows the measured remaining energy using relayed transmission compared to the amount using normal event transmission. However the amount of saved energy gets smaller the more devices are used in the simulation. This can be explained by the fact that a large number of devices in such a small area will also cause an increase in Communication overhead when searching for clusters. This fact is illustrated by Figure 7.2a. The figure shows the average number of messages, that are not related to event transmission, a device has to process. As we can see the amount of those messages will increase with the number of devices in the system.

Figures 7.1c through 7.1f show that while our algorithm still saves energy for an area of 500 square meters, the remaining energy of mobile devices decreases overall when we increase the simulation space. This can be accredited to the fact that due to the increasing space, mobile devices are able to move round more freely and clusters are harder to maintain.

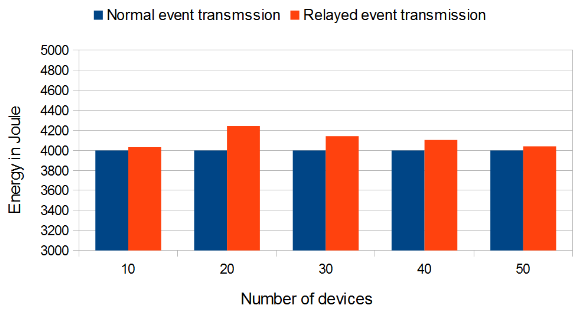
This fact is described by Figure 7.2b which shows the average number of events, a device sends as a Cluster-Member, using WLan instead of the more expensive standards 3G or GSM. As we can see this number decreases decreases steadily for each increase in simulation space.



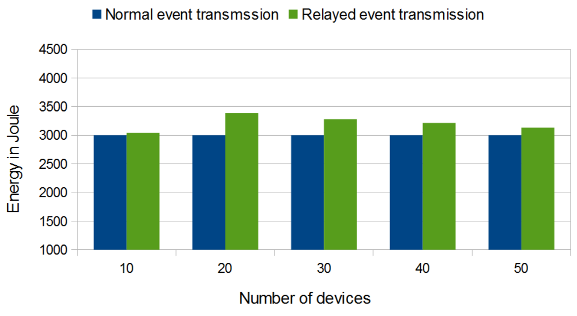
(a) Average remaining device energy using 3G for an area of 250m x 250m



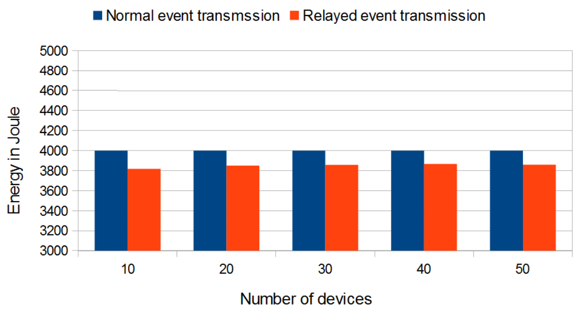
(b) Average remaining device energy using GSM for an area of 250m x 250m



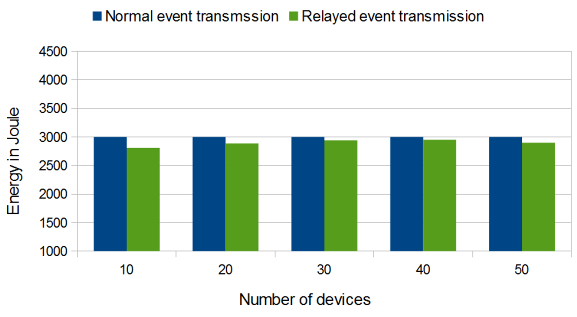
(c) Average remaining device energy using 3G for an area of 500m x 500m



(d) Average remaining device energy using GSM for an area of 500m x 500m

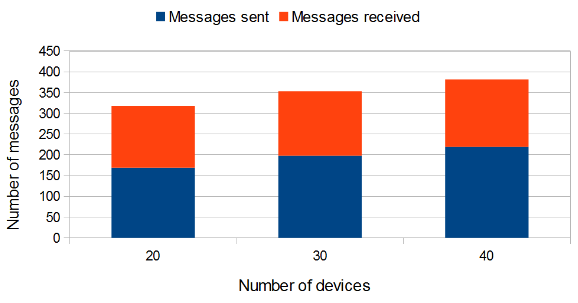


(e) Average remaining device energy using 3G for an area of 750m x 750m

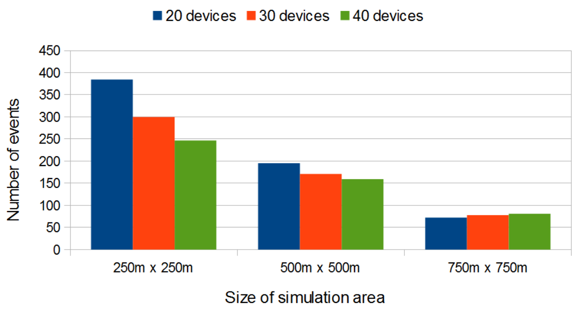


(f) Average remaining device energy using GSM for an area of 750m x 750m

Figure 7.1: Averaged energy measurements for high-frequency event transmission.



(a) Average number of maintenance messages for an area of 250m x 250m



(b) Average number of events per device sent as Cluster-Member

Figure 7.2: Non-Event message counts for high-frequency simulation.

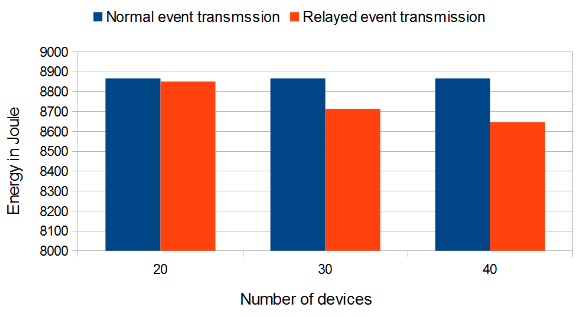
7.2.2 Performance for low-frequency event transmission

Corresponding to the previous subchapter where we compared the performance of our algorithm for high-frequency event transmission, we will now perform the same evaluation for low-frequency event transmission. Events are produced and transmitted every 15 second, which means that every device will send 66 events.

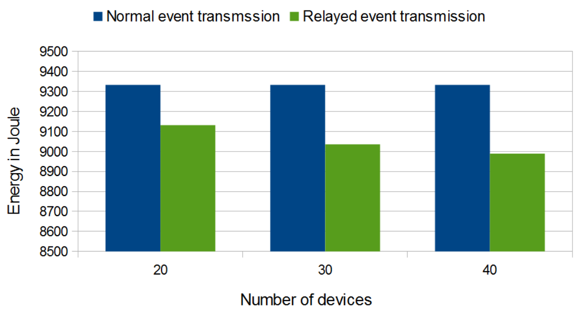
The simulations covered spatial areas of 250 square meters and 500 square meters. Each time the area was sequentially populated by 20, 30 and finally 40 mobile devices. Each combination of area size and number of devices was simulated using the energy model for low-frequency 3G transmissions and using the model for low-frequency GSM.

Figure 7.3 shows the remaining energy using *relayed event transmission* compared to the remaining energy when using normal event transmission. The results show that even for the smallest of our simulated areas, *relayed event transmission* performs worse than normal event transmission. We can explain this with the fact that despite the low event frequency, devices kept searching and performed cluster maintenance at the same intervals as in the high-frequency case, resulting in a noticeable amount of communication overhead. Using the values of Figure 7.2a and our energy model we, we can calculate this overhead for an area of 250 square meters, to be approximately 500 Joules. According to the energy model we used every event transmission via 3G will cost a device 17 Joules. For 66 events this would equal an amount of 1122 Joules. In order to remain profitable the mobile devices in the 250 square meters area would then have to send on average almost every second event message using WLAN instead of 3G. As we can see in Figure 7.4, which shows the average share of events that were sent using WLAN and 3G per device during the simulations for an area of the size of 250 square meters, this is not the case.

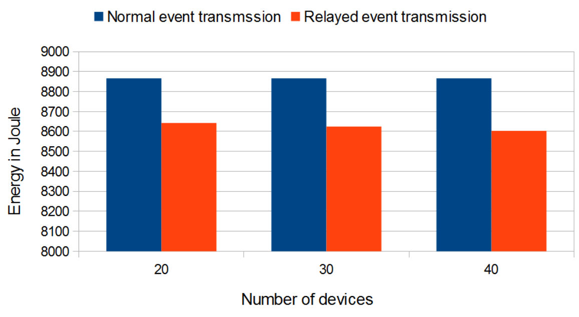
The results for GSM can be explained similarly by performing analogue calculations using the energy model for low-frequency GSM transmissions.



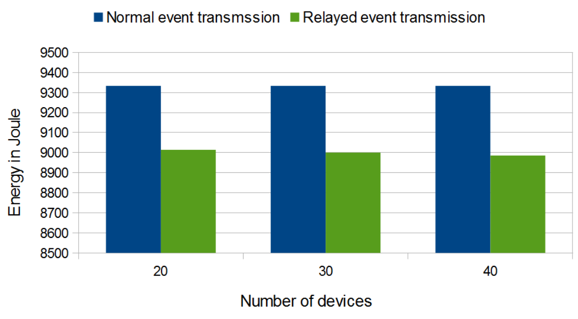
(a) Average remaining device energy using 3G for an area of 250m x 250m



(b) Average remaining device energy using GSM for an area of 250m x 250m



(c) Average remaining device energy using 3G for an area of 500m x 500m



(d) Average remaining device energy using GSM for an area of 500m x 500m

Figure 7.3: Averaged energy measurements for low-frequency event transmission.

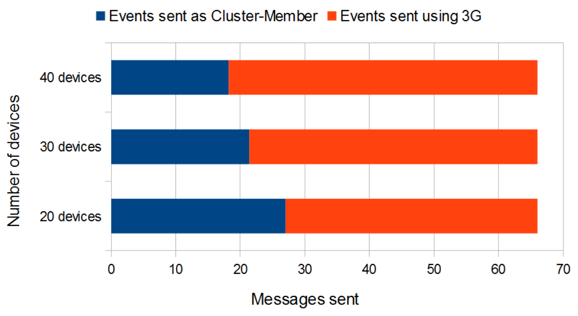
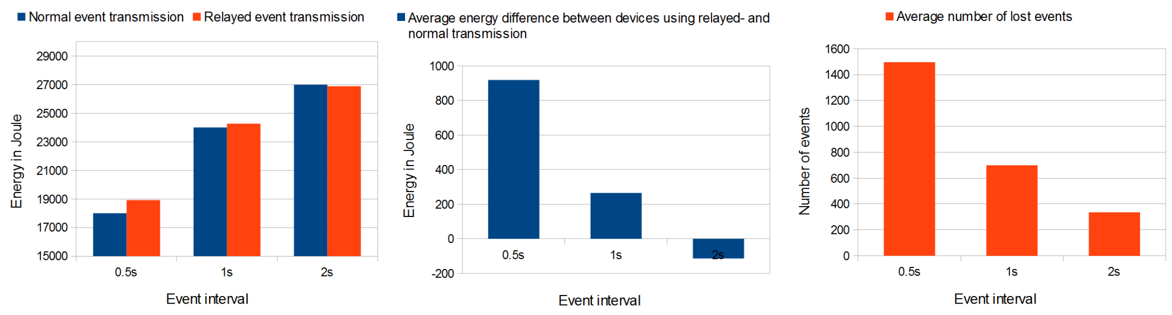


Figure 7.4: Average number of events per device sent as Cluster-Member, in simulations for 250m x 250m

7.2.3 Effects of varying EventIntervals on the system.

The Interval in which events are produced and transmitted obviously has an effect of our system. We examined this effect further by running a set of simulations that produced events at different frequencies and compared the results. The simulation area was fixed at 500 square meters and populated with 20 mobile devices. Each mobile device was initialized with 30000 Joules of power instead of the usual 10000. Events were produced every 0,5 seconds, 1 second and finally 2 seconds.



(a) Average remaining device energy.

(b) Difference between remaining energy of devices.

(c) Average number lost events.

Figure 7.5: Measurements for differing speeds in a 500m x 500m area with 20 devices.

Figure 7.5a shows the average amounts of remaining energy on devices using nomral- and *relayed event transmission*. While relayed transmission performs better at interval of 0.5 and 1 seconds it performs slightly worse than normal transmission at a 2 second interval. With more frequent events, devices in clusters will be able to send more events using the cheaper WLAN communication and hence save more energy in the same amount of time. So it is not surprising that we safe more energy the more frequent the system produces events. Figure 7.5b better illustrates the difference in remaining energies per device from Figure 7.5a.

Since we are transmitting events more frequently we will also lose more events when a Cluster-Member moves out of range of its Cluster-Head. In this case the Cluster-Member will keep sending events to the Cluster-Head until its *RecieveRangeCheckTimer* expires. If we send events with a higher frequency that means we will also send more events during that short period of time, which will never be received. Figure 7.5c displays the average number of lost events relative to event frequency. As we can see we do loose about twice as much events when we double the event frequency.

7.2.4 Effects of varying Speeds on the system.

Since our system incorporates mobile devices we want to examine how the speed in which those mobile devices move will affect it. For this we ran a set of simulation with varying speeds but otherwise fixed parameters. Also we only performed this simulation using 3G communication since the difference between using 3G an GSM exists only in energy costs for transmissions. Events are produced and transmitted every 1 second. The simulation area was fixed at 500 square meters and populated by 20 mobile devices.

To test the effects of different speed on the system we performed simulations for mobile devices with speeds of 5 meters per second, 10 meters per second and 15 meters per second.

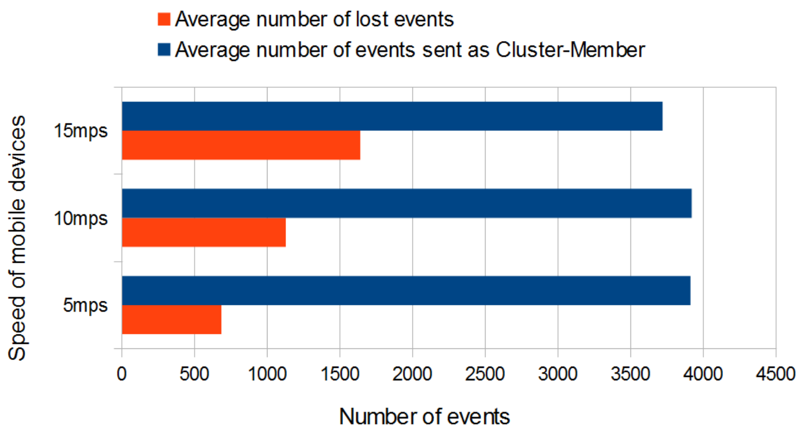
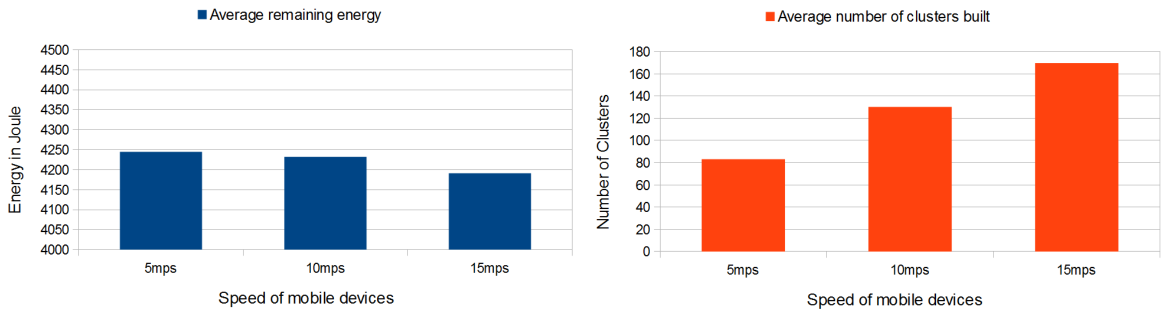


Figure 7.6: Average number of lost events in 500m x 500m area with 20 devices in relation to speed.

Figure 7.7a shows the recorded average values for remaining energy on each mobile device. As we can see the different speed have only little effect on our systems energy consumption. However with faster speeds we do notice an increase in the number of clusters that are formed during the course of the simulation. The number of clusters formed during the simulation is illustrated in Figure 7.7b. This can be explained by the fact, that since the mobile devices are moving faster, Cluster-Head and Cluster-Member devices will also be out of range earlier, when moving away from each other. This leads to the increased number of formed clusters, since clusters will also dissipate earlier. This also explains the increased number of lost event messages with increased speeds, which can be observed in Figure 7.6. The figure shows the average number of lost events next to the overall number of events that were sent as Cluster-Member. Once a device joins a cluster it will send events to the Cluster-Head for at least until its *ReceiveRangeCheckTimer* expires. This timers duration is equal to the parameter *RangeCheckInterval*. The faster a mobile device is out of reach of its Cluster-Head, the more unreceived events it will send until the timer expires.



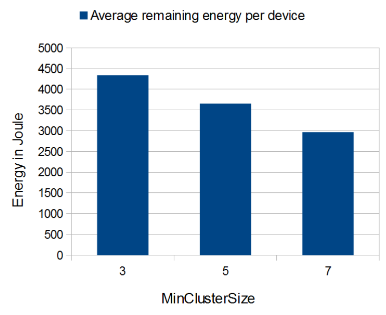
(a) Average remaining device energy using 3G for differing speeds.

(b) Average number of clusters formed during the simulation for differing speeds.

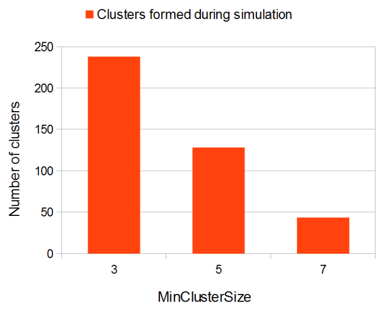
Figure 7.7: Measurements for differing speeds in a 500m x 500m area with 20 devices.

7.2.5 Effects of varying MinClusterSizes on the system.

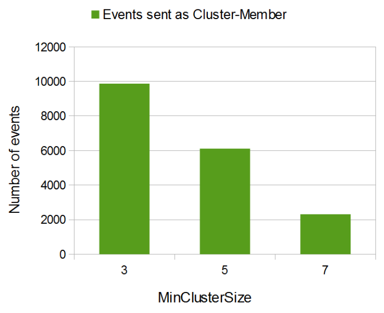
We now want to test what happens to the system if we change the system Parameter *MinClusterSize*. To do so we simulated an area of 250 square meters populated by 40 devices. For this set of simulations we set the parameter *MinClusterSize* sequentially to 3, 5 and 7. We only performed this simulation using 3G communication. Events are produced and transmitted every 1 second.



(a) Average remaining device energy.



(b) Average number of clusters formed during simulation.



(c) Average number of Events sent as Cluster-Member.

Figure 7.8: Measurements for differing speeds in a 500m x 500m area with 20 devices.

Figure 7.8a shows the average remaining device energy for different values of *MinClusterSize*. As we can see, increasing the minimal cluster size results in a decrease in remaining energy. We relate this to the fact that it should gets harder to form new clusters with bigger minimal cluster sizes. This is confirmed by Figure 7.8b, which shows the average number of clusters that were formed during the simulation. It can clearly be seen that this number decreases significantly by increasing the minimal cluster size. This is partly the case because clusters are simply bigger now and devices that were distributed among a number of clusters are now in the same cluster. Figure 7.8c shows the overall number of events sent as Cluster-Member, depending on *MinClusterSize*. According to this figure, far less events are sent inside clusters with higher values for *MinClusterSize*, which indicates that overall less devices find a cluster.

7.2.6 Effects of varying RangeCheckIntervals on the system.

In this subsection we examine the amount of events lost in relation to the system parameter *RangeCheckInterval*. Therefore we simulated our system with RangeCheckInterval set to 5 seconds ,10 seconds and 15 seconds. The simulation was fixed at 500 square meters and populated by 20 mobile devices. Events were produced and transmitted every second.

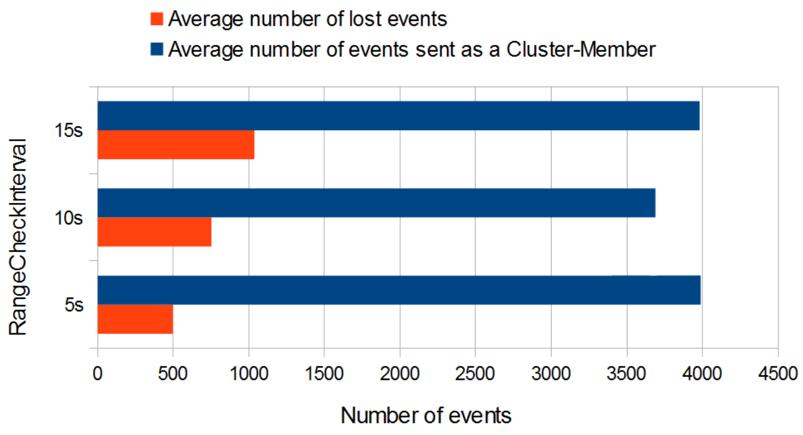


Figure 7.9: Average number of lost events in 500m x 500m area with 20 devices in relation to RangeCheckInterval.

Figure 7.9 shows the results of this set of simulations. The correlation between the size of *RangeCheckInterval* and number of lost events is evident. This makes sense since with increasing intervals between rangechecks, devices have more time to move out of range and transmit events that will never reach the Cluster-Head. The amount of lost messages increases linear to the increase in *RangeCheckInterval*.

7.2.7 Effects of varying HeadCyclePowerAmount on the system.

In order to distribute the additional workload more evenly we introduced the system parameter *HeadCyclePowerAmount*, which limits the time a mobile device acts as Cluster-Head. To check the effectiveness of this parameter we performed a set of simulations with *HeadCyclePowerAmount* set to 100 Joule, 300 Joule and finally 500 Joule. The simulation area was fixed at 500 square meters and populated with 20 devices, each producing events every second.

Figure 7.10 plots all measured values for remaining device energy during the simulations in a sorted manner. This way we can examine the amount of variance between the measured valued. We observe that with smaller values for *HeadCyclePowerAmoun* the skew of the graph becomes smaller. That means that the overall variance in the system becomes smaller, which is exactly what we wanted to accomplish with this system parameter.

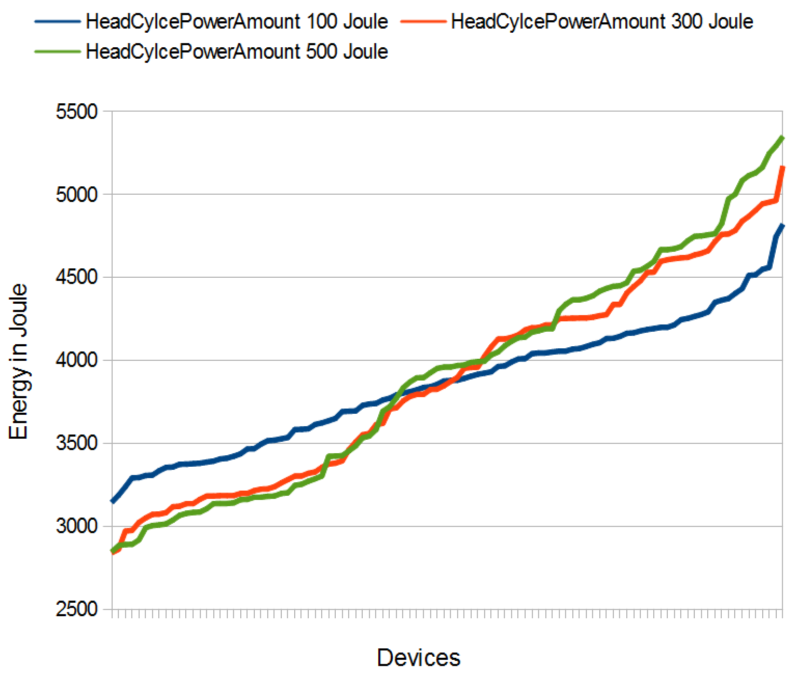


Figure 7.10: Remaining energy measured on all Devices during the simulations sorted by amount to show the amount of variation between values.

8 Related Work

Since CEP systems were first introduced, different approaches for correlation detection have been introduced.

Cordies (Correlation in distributed event services)[KKR10] is a correlation detection language in which complex events are detected using *Correlation Descriptions* (CD). A CD consists of sources, computations, predicates, expressions and events. Sources are simply incoming events. Computations and predicates are defined over event attributes and results of other computations. Each complex event is defined by an expression, which is in turn a logical equation over predicates. If an expression evaluates to *True* a complex event is detected. Predicates and computations can be evaluated as soon as all necessary data is available, which makes correlation detection very efficient since it allows early abortion of the detection process.

Typically systems apply operators directly on incoming events, which the operator obtains through a specific set of event streams. This set of streams is also called an operator's interest. Mobile devices have grown in numbers over the last years and applications with interest in specific spatial ranges are much more common. A driver on the road for example might be interested in the traffic situation ahead of him. Moreover, this range of interest can change with the location of the user. A change in range of interest also means a change in the set of sources for the operator. 'Moving Range Queries in Distributed Complex Event Processing'[KORR12] introduces an algorithm that allows to reconfigure operators, and deal with range of interest changes.

Each consumer has to register a dynamic interest query (diq)= [T,fo,R,δ]. T represents the selected operator, fo stands for a focal object which is used as the center of area of interest, while R denotes the range around fo from which sources are relevant. δ specifies a time-frame in the past for which events are still of interest in the present. Whenever the operator has to be reconfigured the set of sources can be updated by using fo and R to calculate the new range of interest.

To reduce the overhead created by dynamically reconfiguring operators, a partitioned window model for event selection is introduced. A restriction window encases all events relevant for a specific range of interest, while selection windows on event streams contain events that may be considered for the current correlation operation. once a complex event is detected and correlation is finished, selection windows will be shifted so that new events will be available for correlation. Dependencies between events can be modelled by selection window behaviour.

Especially in large scale distributed systems it may be of great benefit to place operators used for complex event detection on different physical network nodes. This way a system might be able to reduce network load. Cordies supports this idea by grouping predicates based on selectiveness and computational overhead. Using these groups the original CD is then split into a subset of CDs. These new CDs can be deployed on separate network nodes using the placement algorithm provided by Cordies. By evaluating very selective CDs first network load will be kept at a minimum.

Systems that apply operators on events often use operator trees to split the complex event detection process. Leaf nodes of an operator tree are the sources, while the higher levels consist of operators, that eventually produce a complex event at the root level. Such a tree can easily be divided by sequentially splitting of sub-trees, which can then be placed on separate nodes.

'Solving the Multi-operator Placement Problem in Large-Scale Operator Networks'[RDR10] is a recent work about operator placement in large-scale distributed network. Goal of the introduced algorithm is to optimize network usage in order to increase scalability.

In order to solve the the *Multi-operator Placement problem* (MOP) it is sufficient to solve the *Single-operator Placement problem* (SOP) for each operator. The SOP considers placement of an operator regarding its neighbours to minimize network usage.

The SOP corresponds to the *Weber Problem*[CT90]. The function for network usage, based on positioning of the operator, is known to be convex. Using the gradient method to approximate the optimal solution in a finite number steps works well.

In order to solve the MOP all operators are initially placed in a centralized manner. Each operator will then possibly be migrated after solving its SOP for the current placement of all operators. After several Iterations of solving each operator's SOP and possible migration, the system will be in an approximately optimal state. In order to deal with changes in the network each operator will re-evaluate its position again after changes in data rate of a connected link or if a neighbour migrates.

In mobile scenarios operator placement presents additional challenges since moving sources and consumers may force the operator to migrate a lot. Operator migration may however be costly due to the need to also migrate event data associate with the operator and. To improve network utilization 'MigCEP: Operator Migration for Mobility Driven Distributed Complex Event Processing' [OKRR13] proposes a plan-based migration approach that predicts movements migrates operators accordingly ahead of time.

Depending on predicted movement MigCEP selects a number of eligible brokers for future placement. After defining a start and endpoint, when the operator has to be available at a new placement the algorithm then produces a *time-graph*. This graph estimates migration cost and time for all possible migration routes over a number of time intervals, using the eligible brokers. Each state corresponds to a certain placement while the edges between the states indicate the migration cost. Finally the k cheapest migration routes are chosen and executed.

Since all values used in the algorithm are estimates, new migration plans are calculated regularly and compared to current plans.

Closest to this work is probably the bachelor thesis 'Ereigniskorrelation auf energiebeschränkten mobilen Endgeräten' [Vet12], which takes a closer look at the energy efficiency of executing certain types of operators directly on mobile devices compared to executing them in an infrastructure after transmitting source data. Operators are then classified using the number of incoming event streams, computational cost and execution frequency. Results indicate that it is more energy efficient to execute operators with few incoming event streams, low computational cost and high execution frequency on the mobile device.

All of the works mentioned above, except the last, work on events that have already been transmitted to the infrastructure. While the last method does include communication with an entry server of an infrastructure, it does not focus on event collection. The operator used in [Vet12] only uses events that are produced by the mobile device itself, not from multiple sources. Also none of the methods above investigates the possibility to split operators further and pre-compute partial results early on mobile devices.

9 Conclusion

The popularity of CEP systems has led to a lot of research on how to optimize event correlation. The rising availabilty of data provided by mobile devices like smart-phones, also brought mobile CEP systems with changing spatial areas of interest into focus. Most research however focuses on optimizing correlation of events that are already in an infrastructure, for example by optimizing operator placement.

In this work we focused on event transmission from mobile devices, acting as sources, to such an infrastructure. We realized this first-hop communication by taking advantage of the mobile devices' ability to communicate between each other using Wireless Lan. Instead of sending events directly to the infrastructure, mobile devices collect events locally using Wireless Lan, before sending them to the server in one message. These transmissions to the server are realized by using the more expensive communication methods 3G or GSM. Since collecting events might result in large messages, we also decided to pre-process the events on the mobile devices. We did so by placing operators on them, which presented special challenges to the operators placed in the infrastructure.

Addressing these problems, we provided an examination and classification of operators in CEP systems regarding their compatibility towards pre-processing. Additionally we presented the algorithms necessary to realize our approach and evaluated them using the OMNeT++ simulation environment. Evaluation showed that for systems that produce events at a high frequency we are able to save energy when using our solution to transmit events to the infrastructure.

Future Work

For future projects we propose the addition of GPS usage to our approach. In this work we did not include GPS data since we could not guarantee that devices would possess a GPS sensor or that they have it turned own. Given that our approach heavily relies on spatial proximity, the use of GPS to find devices in range for Wireless Lan transmissions seems promising. Using GPS we could also try to predict movement patterns and group devices moving in similar directions.

Another possible approach could be to make use of a stationary server that could collect and process GPS data and assign groups, in which events are collected using Wireless Lan, to devices based on its superior knowledge. A server could also divide the area the sources are placed in into cells, in which devices are able to communicate using Wireless Lan, and inform the mobile devices of other devices in their cells.

Bibliography

[BBV09] Balasubramanian, Niranjan ; Balasubramanian, Aruna ; Venkataramani, Arun: Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference Pages 280-293*, 2009 (Cited on pages 5, 12, 13 and 17)

[CH10] Carroll, Aaron ; Heiser, Gernot: An Analysis of Power Consumption in a Smartphone. In: *Proceedings of the 2010 USENIX Annual Technical Conference*, 2010 (Cited on page 12)

[CT90] Chandrasekaran, R. ; Tamir, Arie: Algebraic Optimization: The Fermat-Weber Location Problem. In: *Math. Program.* 46 (1990), S. 219–224 (Cited on page 57)

[KKR10] Koch, Gerald G. ; Koldehofe, Boris ; Rothermel, Kurt: Cordies: expressive event correlation in distributed systems. In: *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems Pages 26-37*, 2010 (Cited on pages 10 and 56)

[KORR12] Koldehofe, Boris ; Ottenwälder, Beate ; Rothermel, Kurt ; Ramachandran, Umakishore: Moving range queries in distributed complex event processing. In: *DEBS*, ACM, 2012. – ISBN 978–1–4503–1315–5, S. 201–212 (Cited on pages 9, 10, 15 and 56)

[LV95] Luckham, David C. ; Vera, James: An Event-Based Architecture Definition Language. In: *IEEE Transactions on Software Engineering* 21 (1995), Nr. 9, S. 717–734 (Cited on page 9)

[OKRR13] Ottenwälder, Beate ; Koldehofe, Boris ; Rothermel, Kurt ; Ramachandran, Umakishore: MigCEP: operator migration for mobility driven distributed complex event processing. In: Chakravarthy, Sharma (Hrsg.) ; Urban, Susan D. (Hrsg.) ; Pietzuch, Peter (Hrsg.) ; Rundensteiner, Elke A. (Hrsg.): *DEBS*, ACM, 2013. – ISBN 978–1–4503–1758–0, 183-194 (Cited on pages 10 and 57)

[PW99] Perkins, Charles E. ; Wang, Kuang-Yeh: Optimized Smooth Handoffs in Mobile IP. In: *ISCC*, IEEE Computer Society, 1999. – ISBN 0–7695–0250–4, 340-346 (Cited on page 46)

[RDR10] Rizou, Stamatia ; Dürr, Frank ; Rothermel, Kurt: Solving the Multi-Operator Placement Problem in Large-Scale Operator Networks. In: *ICCCN*, IEEE, 2010. – ISBN 978–1–4244–7115–7, 1-6 (Cited on pages 10 and 57)

[RZ07] Rahmati, Ahmad ; Zhong, Lin: Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In: *MobiSys*, ACM, 2007. – ISBN 978–1–59593–614–1, 165-178 (Cited on page 17)

[Sch03] Schiller, Jochen: *Mobilkommunikation*. 2. Addison-Wesley, 2003 (Cited on page 11)

[TKK+10] Tariq, Muhammad A. ; Koch, Gerald G. ; Koldehofe, Boris ; Khan, Imran ; Rothermel, Kurt: Dynamic Publish/Subscribe to Meet Subscriber-Defined Delay and Bandwidth Constraints. In: D'Ambra, Pasqua (Hrsg.) ; Guarracino, Mario R. (Hrsg.) ; Talia, Domenico (Hrsg.): *Euro-Par (1)* Bd. 6271, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–15276–4, 458-470 (Cited on page 15)

[Var01] Varga, Andrï¿$\frac{1}{2}$s: The OMNeT++ Discrete Event Simulation System. In: *Proceedings of the European Simulation Multiconference (ESM'2001)* (2001), June (Cited on page 46)

[Vet12] Vetter, Markus: *Ereigniskorrelation auf energiebeschränkten mobilen Endgeräten*. 2012 (Cited on pages 7, 19, 57 and 58)

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Stefan Schmidhäuser)