Protocols for the Efficient Dissemination of Context-Aware Messages

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines Doktors der
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Lars Christian Geiger

aus Nürtingen

Hauptberichter:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Mitberichter:	Prof. Dr. rer. nat. Jörg Hähner
Tag der mündlichen Prüfung:	19. Juli 2016

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart

2016

Acknowledgments

First of all, I would like to thank my doctoral advisor, Prof. Kurt Rothermel, for the opportunity to work on this challenging topic in his research group. Without his guidance, support, and many interesting discussions over the years, this dissertation would not have been possible. Furthermore, my thanks go to the co-reviewer of my thesis, Prof. Jörg Hähner, for taking the time to read this document and for his feedback.

I also want to mention some of the people that I had the privilege to meet and work with during my time at the Distributed Systems group and the Nexus project. Especially Frank Dürr always took the time to discuss my research, sometimes to the detriment of his own full schedule. I would also like to thank (in alphabetical order) Andreas Brodt, Nazario Cipriani, Dominique Dudkowski, Andreas Grau, Ralph Lange, Faraz Memon, Stamatia Rizou, and Harald Weinschrott for the inspiring discussions and the fun I had during my time as a doctoral researcher. Also, I am grateful to Annemarie Rösler, Martina Guttroff, Sabine Thielmann, and Martin Brodbeck for supporting me with administrative duties and thus allowing me to focus on my research. Additionally, during my time at the Distributed Systems group I had the chance to work with a number of talented and enthusiastic students. I greatly enjoyed the opportunity and I am thankful to all of you for that.

I would also like to mention and thank the German Research Foundation, whose funding for the SFB 627 Nexus allowed me to work on my research in the first place.

And last but certainly not least, I would like to thank my parents Werner and Monika, my sister Ramona, and all my closest friends for their support and patience both during my undergrad studies and my doctoral work. They put up with my lack of time and my irritability but hardly ever complained. I am deeply grateful for having all of you in my life.

Contents

Abstract

Context-aware applications are able to react and adapt to the context of their users. This context includes, for instance, location, properties of the user or their surroundings, nearby devices, etc. Over the last years, powerful mobile devices, i.e., smartphones or tablet computers, have become an important part in many people's computing life. Most of these devices maintain a continuous high-speed network connection, allowing to provide distributed applications with an uninterrupted stream of data. Additionally, a huge number of sensors, both in these mobile devices and deployed in our surroundings, enable the creation of comprehensive context models. Such large-scale context models open up new possibilities for the development of context-aware applications by providing access to relevant context information from providers all over the world.

However, until now, applications need to query the context model for relevant information or register for events or messages; it is not possible to "push" information to the mobile devices, neither from the infrastructure nor from other mobile devices. To support application developers, we propose Contextcast, a novel communication paradigm that allows for the dissemination of context-aware (or contextual) messages in a system of context-aware routers. This includes the fundamental semantics to address clients using context constraints and a reference dissemination scheme for such messages.

To enable Contextcast to grow to scales similar to the context-aware systems that it is intended to be used with, we also propose a couple of optimized routing approaches. They are designed to reduce the number of maintenance messages that are necessary for the dissemination of contextual messages. One optimized routing algorithm uses coarse context information to reduce the amount of context updates propagated to routers. To this end, routers use the similarity of contexts to automatically find groups of similar clients, whose information can then be propagated as a single, coarse context. While this reduces the amount of context information to be propagated, the resulting information loss causes more messages to be forwarded, since routers no longer possess exact information to match against the constraints in contextual messages. A configurable similarity threshold allows for various trade-offs between the coarseness of the context information and the resulting additional message load. The second orthogonal routing approach relies on statistics to determine the characteristics of contexts and messages in the system.

Without context knowledge, routers must assume the presence of a matching recipient and forward a message speculatively to disseminate it to all recipients. Using statistics, routers can determine how often certain messages occur and then calculate the benefit of propagating contexts corresponding to these messages. Several parameters enable an administrator to adjust how fast the system reacts to changes, depending on the observed messages and context updates.

Additionally, temporal support extends Contextcast with a powerful mechanism that allows application developers and clients to address messages to certain contexts in the past or future. This includes an additional context attribute *time* and a constraint with various, easy to use temporal operators. We also propose efficient routing approaches for historical and future messages. Routing historical messages focuses on efficient routing while effectively protecting the clients' privacy, i.e., their respective context history. The routing approach for future messages delays forwarding messages until a matching context is registered, thus preventing needlessly forwarded messages.

Zusammenfassung

Kontextbezogene Anwendungen sind in der Lage, auf den Kontext ihrer Nutzer zu reagieren und sich entsprechend anzupassen. Dieser Kontext umfasst beispielsweise den Ort, Eigenschaften der Nutzer oder ihrer Umgebung, Geräte in der Nähe, etc. In jüngerer Zeit wurden leistungsstarke mobile Geräte ein immer wichtiger Aspekt der Computernutzung vieler Menschen. Die meisten dieser Geräte verfügen dabei dauerhaft über eine schnelle Verbindung mit dem Internet. Diese versorgt Verteilte Anwendungen mit einem kontinuierlichen Strom an Daten. Zusätzlich erlaubt eine große Anzahl an Sensoren, sowohl in mobilen Geräten als auch in unserer Umgebung, die Erstellung von umfassenden Kontextmodellen. Solche groß angelegten Kontextmodelle erlauben den Zugriff auf Kontextinformationen von Anbietern auf der ganzen Welt. Dadurch ermöglichen sie neue Möglichkeiten für die Entwicklung kontextbezogener Anwendungen.

Bis jetzt müssen Anwendungen relevante Informationen aus dem Kontextmodell abrufen oder sich für Ereignisse oder Nachrichten registrieren; es ist nicht möglich Informationen mittels eines “Push”-Verfahrens zu den mobilen Geräten zu übermitteln, weder aus der Infrastruktur noch von anderen mobilen Geräten. Zur Unterstützung mobiler Anwendungen und deren Entwickler schlagen wir Contextcast vor, ein neuartiges Kommunikationsparadigma, das die Verbreitung kontextbezogener Nachrichten mit Hilfe von kontextbezogenen Routern ermöglicht. Dies umfasst die grundlegende Semantik um Nutzer zu adressieren sowie einen Referenz-Verteilalgorithmus für solche Nachrichten.

Um Contextcast in ähnliche Regionen skalieren zu können wie die kontextbezogenen Systeme, in denen es genutzt werden soll, schlagen wir außerdem eine Reihe optimierter Vermittlungsverfahren vor. Sie sind darauf ausgelegt die Anzahl der Verwaltungsnachrichten in Zusammenhang mit der Vermittlung kontextbezogener Nachrichten zu reduzieren. Einer dieser optimierten Vermittlungsalgorithmen nutzt grobe Kontextinformation um die Menge an Kontextupdates, die zu Routern propagiert werden müssen, zu reduzieren. Zu diesem Zweck nutzen Router die Ähnlichkeit von Kontexten um automatisch Gruppen ähnlicher Nutzer zu erkennen, deren Information dann als einzelner, grober Kontext propagiert werden kann. Obwohl dadurch die Menge an propagierter Kontextinformation verringert wird, müssen auf Grund des Informationsverlusts mehr Nachrichten übermittelt werden: Router besitzen keine exakten Informationen mehr, die sie für ihre Weiterleitungsent-

scheidung gegen die Prädikate in der Adressierung kontextbezogener Nachrichten auswerten können. Eine konfigurierbare Ähnlichkeitsschwelle erlaubt die Grobheit der Kontextinformation und die daraus resultierende zusätzliche Nachrichtenlast gegeneinander abzuwägen. Der zweite, orthogonale Ansatz verwendet Statistiken um die Charakteristika von Kontexten und Nachrichten im System zu ermitteln. Ohne Kontextwissen müssen Router die Anwesenheit eine passenden Empfängers annehmen und eine Nachricht spekulativ weiterleiten, um alle Empfänger zu erreichen. Mit Hilfe der Statistiken kann ein Router bestimmen, wie oft bestimmte Nachrichten auftreten und daraus den Nutzen bestimmen, den die Propagation von Kontexten zu diesen Nachrichten hätte. Mehrere Parameter ermöglichen einem Administrator einzustellen, wie schnell das System auf Veränderungen reagiert, abhängig von den beobachteten Nachrichten und Kontextupdates.

Darüber hinaus erweitert eine temporale Unterstützung Contextcast um einen mächtigen Mechanismus, der es Anwendungsentwicklern und Nutzern erlaubt, Nachrichten an Kontexte in der Vergangenheit oder Zukunft zu adressieren. Dies beinhaltet ein Attribute *time*, sowie ein entsprechendes Prädikat mit einer ganzen Reihe einfach zu verwendender temporaler Operatoren. Wir schlagen auch effiziente Verteilverfahren für historische und zukünftige Nachrichten vor. Die Verteilung historischer Nachrichten hat ihren Schwerpunkt dabei auf der effizienten Verteilung, die gleichzeitig effektiv die Privatsphäre der Nutzer schützt, d.h. ihren Kontextverlauf. Der Ansatz zur Verteilung zukünftiger Nachrichten verzögert solche Nachrichten, bis ein passender Kontext registriert wird und verhindert so die Verteilung von Nachrichten, für die sich nie ein passender Kontext anmeldet.

Chapter 1

Introduction

> A journey of a thousand miles begins with a single step.
>
> *(Laozi)*

1.1 Motivation

Manhattan, downtown, September 15th 2016, during the evening rush hour. Tens of thousands of people are on their way home after work or are going out. Most of them carry a powerful mobile device, all equipped with previously unimaginable computing power and numerous sensors that constantly gather information about their vicinity. All maintain a high-speed connection to the Internet, either via cellular connections or wireless access points scattered throughout the city. Via this connection, the devices are constantly exchanging information with a huge, distributed context model, jointly operated by Internet Service Providers (ISPs), various other companies, Nonprofit Organizations (NPOs), and private individuals. Invisibly working in the background, numerous applications process this data to provide valuable services and information to the device owners.

Suddenly, a driver loses control of his vehicles on the Brooklyn Bridge, crashing into two other motorists before finally coming to a stop. The lanes leading to Brooklyn are blocked after the crash. This causes the streets in the financial district to become clogged by all the people heading out of Manhattan. Within minutes, the traffic in this part of the city would come to a halt.

A traffic control system is constantly monitoring and analyzing the enormous amount of data available in the context model; a continuous query provides it with a stream of anonymous motion patterns from people in the city. Within seconds, it matches the sudden slowdown in the area to a traffic jam situation, pinpointing the problem to the Brooklyn Bridge. A context-aware workflow is set in motion, which notifies a human operator about the problem since it occurred during rush hour.

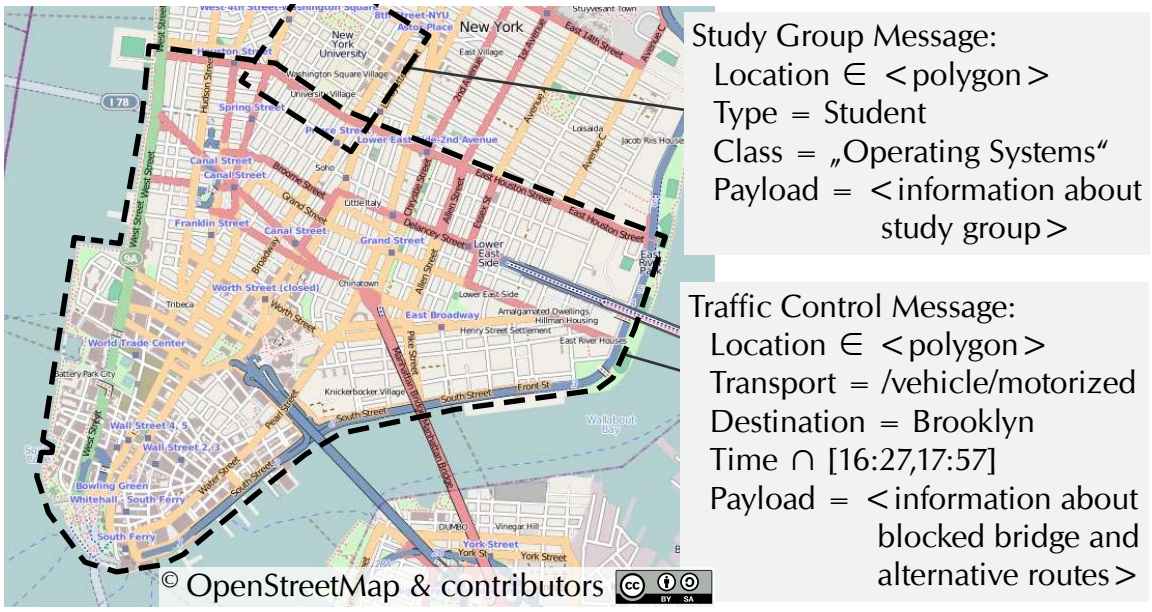


Figure 1.1: Contextcast messages in downtown Manhattan

After that, it starts directing traffic away from the Brooklyn Bridge to alleviate the beginning traffic jam.

The traffic control system sends out a context-aware message (or a *Contextcast*) to inform people in the area of the problem and alternate routes. The message is addressed to all people in the area south of Houston street, driving a motor vehicle, with a destination somewhere in Brooklyn. Its payload contains a traffic warning for Brooklyn Bridge and directs drivers (or their navigation systems) to use either the Manhattan Bridge or the Williamsburg Bridge to get to Brooklyn (cf. Figure 1.1). The mobile devices transfer the information into the on-board computer of the car and therefore into the satellite navigation system, which then offers the driver an alternative route.

Additionally, the experienced operator has estimated that a crash of this severity takes at least an hour to clear and has thus set the validity time of the message from now to 90 minutes into the future. Thus, anybody entering the target area in these 90 minutes or getting into their car to drive over the Brooklyn Bridge also receives the message once their context changes and matches the message.

At the same time, just a little bit further north, a group of students at New York University is preparing for an exam the following day with a late night study session. They send out a message to all students close to the campus, who attended the same class, to join them for the study session (see Figure 1.1). This uses a different kind of context, the information about the classes a student attends, which is part of a student's social network profile. Thus, the same Contextcast system scales from a

small, local message dissemination on a university campus to a large scale system with several hundred thousand possible recipients in downtown Manhattan.

These scenarios show how the rapid progress in various areas such as mobile computing and communication, as well as the spreading of context-aware computing contributes to novel and useful services in the future. In contrast to current techniques such as (IP-based) Multicast or Publish/subscribe, addressing clients via their contexts frees users from the burden of explicitly selecting interesting messages by joining or subscribing. Often, this is even impossible, since it may not be known in advance what kind of messages and what information are going to occur in a network.

However, missing pieces to enable such a large-scale context-aware communication are (1) a method that allows senders to address recipients using constraints on their context attributes, and (2) efficient dissemination algorithms to deliver such context-aware messages. A communication system designed using these building blocks offers a valuable service for context-aware applications on a large scale.

This dissertation focuses on the development of these mechanisms and the design of a context-aware communication system for use in (but not limited to) large-scale context-aware system such as Nexus (see Section 1.2.3). In this chapter, we provide an overview of the background for and the challenges addressed in this dissertation.

First, we discuss the background and related technologies in more detail in Section 1.2. Second, Section 1.3 shows the focus and contributions of this dissertation. Finally, in Section 1.4, we outline the contents of the following chapters, in which we present the results of this dissertation.

1.2 Background

1.2.1 Technological Trends

The development of a Contextcast system has been supported mainly by two ongoing technological trends: *miniaturization* and *wide availability of mobile Internet connectivity*.

For almost half a century now, Moore's Law [Moo65] has closely approximated the development of integrated circuits. This has led to the wide availability of ever smaller and more powerful mobile devices, including notebook computers, tablet computers, or smartphones. At the same time, these devices have gotten more and more abilities to sense their surroundings. A modern smartphone comes equipped with a Global Positioning System (GPS) sensor, acceleration and gyration sensors, light sensors, microphones, etc. All of these enable a device to capture context information about its owner and use the gathered data for context-aware applications.

Additionally, all such mobile devices include technology to establish a high-speed connection to the Internet. Cellular network connections, such as the 3rd generation Universal Mobile Telecommunications System (UMTS) are widely available today and offer bandwidths of several hundred kbit/s and up to 7.2 Mbit/s with High-Speed Downlink Packet Access (HSDPA). Its successor, Long Term Evolution (LTE), will further raise this towards peak downlink rates of 100 Mbit/s. Another positive influence is that prices for data transferred via cellular connections has dropped dramatically in recent years, thus allowing people to maintain a continuous Internet connection, independent of their current location. In certain areas—mostly public places such as airports or cafés—Wireless Local Area Network (WLAN) hotspots are available, which provide a faster and often even cheaper alternative. The IEEE 802.11n standard, successor to 802.11a and 802.11g, allows data transfer rates of up to 450 Mbit/s and is seeing adoption in newer devices, while devices equipped with one of its predecessors are in wide use today. Therefore, more and more people possess at least one device which maintains a more or less continuous, high-speed connection to the Internet.

1.2.2 Paradigmatic Trend: Context-aware Computing

A continuous connection to the Internet provides users and applications with access to various information gathered by sensors in our surroundings. This, combined with the information captured by small personal devices, has given rise to a new class of applications: context-aware systems. Context-aware systems improve the user experience, e.g., by presenting relevant information or by executing certain tasks without explicit interaction.

There exist various definitions for the terms "context" and "context-aware"; the predominant one states that "*Context* is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." And "[a *context-aware* system] uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." [DA99] In particular, this definition of context includes the user's location. Context-aware systems can therefore be viewed as a natural extension of Location-based Services (LBSs), which use only the current location to provide relevant information and services.

To provide their functionality, context-aware applications access the relevant information in a context model. Existing context models, however, use their own formats and communication mechanisms [BDR07] or provide information for a specific application only [BCQ+07]. One reason is that the complexity of managing such models increases with the scope (local vs. global) and the level of detail (coarse

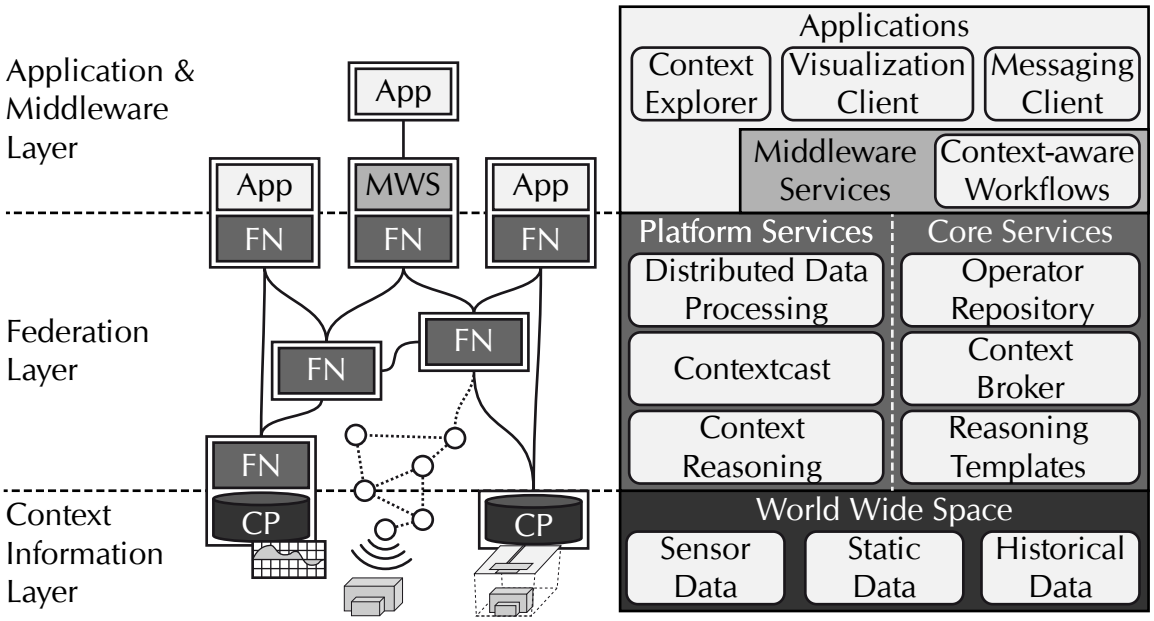


Figure 1.2: The Nexus architecture [LCG+09]

vs. fine). This fragmentation limits the reuse of information for different context-aware applications and causes information and functionality to be duplicated in different models.

1.2.3 SFB 627: Nexus

As we have stated in the previous section, context-aware applications need to model context to provide their functionality. Application-specific, limited models, however, are inefficient due to the necessary duplication of functionality and information. The collaborative research center[1] 627 "Nexus", funded by the Deutsche Forschungsgemeinschaft (DFG), developed technologies to provide a detailed, large-scale context model [GBH+05, REF+06]. The project deals with the modeling of and efficient access to global, fine grained context information for different context-aware applications.

At the core of the project lies the Nexus architecture, which provides a detailed, large-scale context model. It is shown in Figure 1.2 and described in more detail in [LCG+09]. The system is divided into three main layers (or tiers), which we present here in a short overview: *Context Information Layer*, *Federation Layer*, and *Application & Middleware Layer*.

[1] German: Sonderforschungsbereich (SFB)

- The *Context Information Layer* contains Context Providers (CPs) from different data providers. They provide access to various forms of context information, such as static, sensed, or historical data. Each CP usually only contains some part of the global model, e.g., a company can provide detailed floor plans and sensor information for their building(s), while another company provides global map data, but much less detailed.
- The *Federation Layer* in the middle provides various services. These are usually distributed among the available Federation Nodes (FNs). We can distinguish two types of services: Platform Services, which are typically used by applications, such as Contextcast or Context Reasoning, and Core Services, which provide functionality for Platform Services, such as the Context Broker [LDR10], a distributed system to discover relevant context providers for query processing.
- The *Application & Middleware Layer* at the top primarily hosts the various context-aware applications. They can access the services provided by the Federation Layer to Additionally, Nexus provides support for Context-aware Workflows as part of its Middleware Services (MWSs). This allows designers to split off parts of an application's logic into one or more context-aware workflows.

Among the services in the Federation Layer of the Nexus platform is a context-based communication mechanism (or Contextcast). It allows messages addressed to clients with a specific context and handles the efficient dissemination of these messages. This service and its efficient implementation are the main topics of this dissertation.

1.2.4 Context-aware Communication

Context-aware communication, or Contextcast, provides context-aware applications with the means to disseminate messages to a group of recipients with a certain context. This enables senders to address otherwise unknown recipients. In particular, Contextcast does not require explicit addresses, such as Internet Protocol (IP) addresses. In this regard, Contextcast differs from other communication forms, which employ explicit (group) addresses for their messages. Instead, contextual messages[2] specify constraints on the context attributes of clients. Messages are delivered to the clients whose context attributes match these constraints. A network of context-aware

[2]We use the terms "contextual messages" and "context-aware messages" interchangeably throughout this dissertation.

routers is responsible for the efficient dissemination of the contextual messages to their recipients.

A typical scenario for this new communication paradigm is the dissemination of warning messages. This includes warnings about high pollen counts in an area for people with particular allergies or the traffic warning from Section 1.1. The Contextcast system is responsible for the delivery to the motorists in an area with a particular destination. Clients automatically receive the messages, without having to constantly query the system for information. Other possible uses include information about cultural events, appropriate for a person's interest and schedule, or advertisement messages, for people in the vicinity of a store that match the store's customer profile.

In addition to these informative, unidirectional messages, which are sent out by organizations or companies, another possible use for Contextcast is as a basis for a context-aware instant messaging system (see, e.g., [DPG+08]). Such a system provides clients with a bidirectional, context-aware messaging. This can be used, for instance, for the example from Section 1.1, to invite people that are currently on campus to a study group matching their academic major and semester.

Contextcast has some obvious similarities to other communication schemes but improves upon them in various ways. We give a short overview of the state of the art here and provide a much more detailed classification and discussion of related work in Section 2.3.

Related Communication Mechanisms

Group Communication (or multicast). This communication paradigm allows one or more senders to address a message to a group of recipients (hence the name). The group is specified using a group address or identifier. Recipients explicitly join the group(s) of messages they wish to receive. Routers efficiently disseminate messages to all group members using a dissemination tree; messages are only duplicated at branching points in the tree, thus minimizing the message load.

This technique has two drawbacks compared to Contextcast: First, it requires clients to join the group(s) they are interested in. Providing a context-aware communication system on this basis would entail a knowledge of all groups and a specification of the context that each one represents. Only then can clients join and leave groups depending on their current context. Contextcast, in contrast, requires no action on the part of the recipients; it provides senders with a very expressive way to specify recipients in terms of their context. Second, and related to the previous point, specifying recipients using their context provides a very fine-grained addressing scheme. A message can address a single client or all the clients in a country. Even if it were possible to automatically map recipients to one or more

groups according to their context, this would not scale to the same fine granularity. Replicating such a fine-grained addressing using groups would require a group for every element of the power set of all clients (except the empty set and the set of all clients). This is obviously not scalable beyond a few participants.

Geographical Communication (or geocast). Geocast also allows addressing and disseminating messages to a group of recipients. The groups, however, are implicitly formed by the presence of clients inside of the target area of the message. Similarly to multicast, the actual message dissemination usually takes place along a tree of routers to minimize the overhead of duplicate messages.

Arbitrary geometrical shapes as target area allow for a very fine-grained specification of recipient groups. The granularity of the addressing scheme is limited only by the precision of the positioning system used to determine client location. Symbolic addresses, such as "building V.38, room 2.308", may not be as fine-grained but correspond more closely to the human perception of areas.

While geocast offers a fine-grained addressing of clients according to their location and an efficient dissemination of messages, it lacks the ability to address other context attributes. Also, the message dissemination of geocast systems commonly relies on the containment relation present in locations. Since arbitrary context attributes do not satisfy this property, the scheme cannot be extended easily to support such attributes.

Event-based Systems (or Publish/subscribe). Publish/subscribe (Pub/sub) systems, especially the content-based variety, provides consumers with an expressive way to specify and receive interesting events based on the events' content. As such, it is very similar to the way that contextual messages are addressed using constraints on context attributes. We therefore model our context-aware routing approach after the concepts of content-based Pub/sub.

However, content-based Pub/sub systems require clients to subscribe to interesting events. Translating arbitrary context attributes into filters for event content is difficult to impossible. Contextcast, in contrast, lets senders specify the intended recipients using their context attributes. It thus removes the complications that arise from mapping context attributes to event subscriptions.

In its pure form, content-based routing relies solely on client subscriptions on event content and does not consider the recipient context when disseminating events. There are, however, approaches that aim to extend the Pub/sub approach to become context-aware.

First, *Location-dependent subscriptions* allow clients to incorporate their location into subscriptions. The approach does not support client context other than location, though. Second, developed independently from our approach, a *context-aware Pub/sub* system aims to incorporate the context of both producers and consumers. Any node (both producers and consumers) can specify their context: Events can

contain constraints on consumer context in addition to the data of the event; subscriptions can specify constraints on producer context in addition to constraints on event data. While Contextcast also employs constraints on the recipient context, its semantics are simpler and purely sender-centric: only contextual messages contain constraints, while recipients only specify their context, not additional constraints on message content. Despite context-aware Pub/sub providing a similar functionality as Contextcast, we focus on the efficiency of the routing algorithms for contextual messages. This is particularly important for client contexts, which are very dynamic in nature. In addition, Contextcast provides a temporal extension for contextual messages, allowing senders to address clients with a particular context some time in the past or future. The focus of this extension is an efficient as well as privacy-aware dissemination of such messages.

1.3 Focus and Contributions

Following the examples and the discussions in the previous sections, we are now able to specify the focus and contributions of this dissertation.

1.3.1 Focus

Regarding the scope of this dissertation, we can identify a number of main aspects: functionality, system size, system structure, location model, and context quality.

The trends we discussed in the previous section suggest an increasing focus on context-aware computing, freeing user attention for more important tasks. Contextcast provides programmers with the communication *functionality* to develop innovative context-aware applications and services. To this end, Contextcast needs to be both flexible and easy to use to become a useful tool for application developers. Regarding functionality, this dissertation therefore focuses on a communication infrastructure with a relatively simple Application Programming Interface (API) so developers do not need to concern themselves with the actual implementation of the system. Additionally, we introduce an intuitive but powerful addressing mechanism, allowing recipients to be specified using constraints on context attributes. This allows to determine recipients using constraints similar to the informal "people with the major *computer science*" or "people who like jazz music". A temporal addressing scheme further increases this flexibility, allowing the specification of recipients that matched a set of constraints at some time in the past or will match it in the future.

System size refers to the scale of a Contextcast system. The Nexus project (see Section 1.2.3), which provides the surroundings for Contextcast, envisions a future of large-scale, shared context models. The information in these models ranges from

very coarse to very detailed. To support context-aware application and services in a large-scale system such as Nexus requires a communication framework that also scales well to a large number of clients. Thus, we focus on a distributed system of context-aware routers, which can be extended easily to support more clients. Such a system, however, requires scalable routing approaches to deliver a given message to the clients matching its addressing. In particular, the routing algorithms need to keep both the message dissemination and maintenance load low.

System structure is concerned with the network structure of our context-aware communication system. Both currently prevalent mobile data connections, cellular and WLAN, both offer a wireless connection to a fixed, wired network infrastructure. Thus, the focus of this dissertation is on a Contextcast system in such a fixed network infrastructure instead of, e.g., wireless ad-hoc networks. In general, for economical and practical reasons, a global context-aware communication framework should exploit existing infrastructure, such as the Internet. Previous attempts to add functionality on the IP layer, however, have proven to move very slowly, if at all, e.g., the introduction of multicast mechanisms beyond individual ISPs [CRSZ02]. For this reason, this dissertation presents an approach for context-aware communication that does not require the modification of existing routers. Instead, we use an overlay network of context-aware routers to disseminate messages. This functionality can be incorporated in future IP routers, supplementing or replacing nodes in the overlay.

The *location model* plays an important role, due to the special role of *location* as primary context (cf. [RBB03]). We employ *location* to structure the Contextcast system, to exploit local clusterings of similar clients in the physical world. There exist a number of different location models, from geometric models, such as World Geodetic System 84 (WGS84) [Nat97], over symbolic models [Leo98], where certain areas become logical locations, to hybrid models, which combine the two [DR03]. Both symbolic and hybrid models increase the complexity over a simpler geometric model. To limit the complexity in this respect, we have chosen a geometric location model based on WGS84. This model is sufficient to demonstrate the general principles in developing a Contextcast system. Also, it does not limit the system, as a more complex model can be substituted when necessary. (As we are discussing in Chapter 3, all that is required is a definition of the attribute, e.g., a symbolic location model, as well as the supported operators for this attribute, such as a containment operator.)

Naturally, whenever we observe or describe the physical world, certain inaccuracies are present in the data. There are various reasons for those, such as the inaccuracy of a GPS sensor or simply outdated information. While these inaccuracies may result from different sources, we can group them under the term *context quality*. In this dissertation, we handle the aspect of context quality in particular through the development of routing algorithms for imperfect information. In particular, these

algorithms work with coarse, aggregated or incomplete data to lower the overall system load and thus improve Contextcast's scalability.

1.3.2 Contributions

The goal of this dissertation is the design of a context-aware communication system, to provide context-aware systems with a means to disseminate messages using constraints on context attributes. Such a system provides the developers of large-scale context-aware applications and services with a valuable communication platform that does not rely on traditional communication patterns. Due to the scale of such systems, a particular focus of this work is the efficiency and scalability of the developed mechanisms.

The particular contributions can be derived from the focus discussed in the previous section and summarized as follows:

1. We provide a semantic definition for a context-aware communication system that allows for arbitrary context attributes and constraints on these attributes. These constraints allow for a context-aware addressing of messages.

2. We introduce a reference message dissemination algorithm for a distributed system of context-aware routers. This algorithm delivers messages to all those access networks where matching recipients have registered their information. It achieves this using the minimum number of forwarded copies of a message in the given network topology.

3. We present an optimization to the reference dissemination algorithm to improve the scalability of the system. It reduces the number of context updates that are necessary for the maintenance of context information on routers, thus lowering the system load. To this end, it allows routing context-aware messages using coarse context knowledge. This coarse information is automatically derived from client contexts and their similarity.

4. A second, orthogonal improvement of the routing algorithm optimizes the nodes that context information is maintained on. Using statistics, the routers determine the direction where particular message classes originate. Only contexts that may match these messages need to be forwarded in this direction. Again, this reduces the amount of update messages to improve the system scalability.

5. We detail an extension to the basic Contextcast semantics, allowing senders to address messages to recipients with a matching context some time in the past or future. This includes storage and indexing of client contexts, complete with

privacy preserving measures, as well as efficient routing mechanisms for these messages.

6. We provide performance analyses to validate the dissemination algorithms that we introduced for a context-aware communication system.

The author has developed the basic concepts of Contextcast, its semantics and the reference dissemination algorithm (items 1 and 2), as well as the optimized dissemination approaches using coarse context information and adaptively propagating context information (items 3 and 4, respectively). He has also performed an evaluation of these optimized algorithms using a prototype he implemented (item 6). The concepts for adding support for temporal relations to Contextcast have been developed by the author; this work was performed in collaboration with Ronald Schertle, who also performed their evaluation, during his Diplomarbeit (item 5). Throughout the process, Frank Dürr contributed to the refinement of all these approaches via discussions and reviews.

1.4 Structure

The structure of this thesis follows the basic design of the Contextcast system: It starts with the semantic definitions and the reference dissemination algorithms that form the foundation of such a context-aware communication system. After that, it shows two approaches that improve the scalability of the system by reducing the update load that is required to maintain the routing information. Finally, it introduces an extension to the Contextcast paradigm, allowing the addressing of clients with a certain context some time in the past or future. The following provides a more detailed overview of these items.

Chapter 2 presents the necessary foundations for a context-based communication system. We start by introducing the notion of context-based communication, followed by a discussion of the requirements for a large-scale Contextcast system. After that, we give an overview of related work and classify the approaches according to various criteria; in particular, we focus on the improvements that Contextcast offers over other related concepts. Finally, we show the system model that serves as the basis for the remainder of this thesis. Some of these concepts have been presented in a previous paper at the IEEE International Conference on Communications 2009 [GDR09].

In Chapter 3, we introduce the basic, formal semantics of context-aware communication. This includes a definition of the primitive elements of such a system, client contexts and contextual messages, as well as the matching semantics between the two. The latter is particularly important since it specifies how and which contexts

are addressed. After that, we discuss the dissemination semantics in a distributed system of context-aware routers and provide a reference algorithm for a directed dissemination of contextual messages using client context information. We end the chapter with a short discussion of the system load caused by the reference dissemination, in particular the update load caused by maintaining context information on routers. This motivates the advanced routing algorithms we present in the following two chapters. Parts of this work have also been included in [GDR09].

Chapter 4 contains the first of two optimized routing algorithms for Contextcast. It is based on the idea that using coarser information for routing contextual messages allows the routers to reduce the amount of context updates. We introduce this concept using the example of coarser *location* information. While the approach works well for *location* with the inherent discretization of our system design, it falls short for context attributes in general. To overcome this limitation, we show a generalized approach where routers exploit similarities between contexts to aggregate multiple contexts into a single, coarser representation. We also discuss how messages are disseminated with the inherent loss of such coarser context information. For both algorithms, we therefore present simulation results that show the update and message load and compare it to the reference dissemination algorithm. We also present results of how the aggregation affects system load over time, i.e., whether clients connecting and disconnecting leads to a noticeable decrease in efficiency due to the aggregations degenerating. The approach has been introduced in parts in a paper at the 6th Euro-NF Conference on Next Generation Internet (NGI) 2010 [GDR10].

The second advanced routing approach is the topic of Chapter 5. At its core lies the idea that it is unnecessary to maintain every context throughout the router network. If a context is used less often for forwarding a message than it is updated, it is more efficient to simply "flood" such messages—potentially without a matching recipient—instead of constantly updating the context information. To aid in this decision, we present a number of statistics that each router maintains for client contexts and messages. We also show how these statistics can be used to calculate the benefit of a piece of context information, i.e., how many messages it prevents at what cost in update load. The routers then use this benefit to adaptively decide whether a piece of information lowers the overall load on a given link. Again, forwarding messages without context knowledge causes a certain amount of needlessly forwarded message, thus counteracting the reduction achieved by the adaptive propagation of contexts. We show simulation results to highlight the reduced system load that is possible with the adaptive propagation of contexts. Additionally, we show how the system reacts to changes in message addressing, thus requiring a different set of context information for efficient operation. Parts of this work have been included in a previous paper at the 7th International Conference on Wireless

and Mobile Computing, Networking and Communications (WiMob 2011) [GDR11].

Chapter 6 extends our Contextcast system to allow for a temporal addressing of contexts, both historical and future ones. Addressing historical contexts requires some form of context archive, which poses a serious privacy concern. We show how contexts are stored locally in access networks and how Virtual Identities are used to protect the clients' privacy. Additionally, we present a multi-step routing approach to first find matching contexts, then resolve their identities and then deliver a message, either directly or via a mailbox. We also introduce a routing mechanism for future messages, which exploits the propagation of contexts to delay message forwarding until a matching context registers with the system. For historical messages, we present simulation results that show how collecting the local results can significantly improve the load of the historical routing algorithm. For the routing of future messages, we provide an analytical evaluation of the storage requirements for delaying messages. Some concepts have been introduced in a paper at the 6th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2009) [GSDR09].

Finally, Chapter 7 concludes the dissertation with a summary of our contributions and an outlook on promising future research directions.

Chapter 2

Foundations

> A successful man is one who can lay a firm foundation with the bricks others have thrown at him.
>
> *(David Brinkley)*

2.1 Context-based Communication: Contextcast

A context-based communication system—or Contextcast—offers an infrastructure that provides context-aware message dissemination for applications and clients in context-aware systems. In the following paragraphs, we provide an intuitive notion of the concepts, which should suffice as basis for the discussions in this chapter; we formally define contextual messages, contexts, the matching between the two, etc. in Chapter 3.

Contextcast as a communication framework is intended for (but not limited to) use in large-scale context-aware systems. It allows contextual messages to be addressed and disseminated using the recipients' contexts instead of explicit addressing via, e.g., IP addresses or group names/addresses. Applications interact with such a system by registering contexts and sending messages with constraints on clients' contexts. This paradigm offers great flexibility—no need to create particular recipient groups in advance—while at the same time being simpler for users—applications can derive the context of a user themselves or the users can describe it in terms of a number of attributes.

As such, it differs from traditional multicast or Pub/sub approaches, which require joining a recipient group, subscribing to certain events, and/or explicit addressing of recipients by senders. When using Contextcast, an application or a client does not need to join or subscribe, instead they simply register one or more contexts that describe this client; this registration can also happen automatically by deriving a client's context from a set of observed conditions, e.g., by sensors in a smartphone or their surroundings. Similarly, when sending messages, an

application developer or a user can simply use context constraints to specify which clients should receive a particular message. The system is responsible for the delivery of the message. Thus, Contextcast can serve as the binding component between different parts and services in large-scale context-aware applications.

Similarly to Pub/sub systems, the Contextcast paradigm decouples senders and recipients in three main aspects (see [EFGK03]): (1) *space decoupling:* senders and recipients do not need to know each other nor do they even need to know how many senders or recipients, respectively, are involved for a particular type of message, (2) *interaction decoupling:* both sending and receiving messages takes place asynchronously, thus not blocking senders and recipients, and (3) *time decoupling:* through the temporal extension we introduce in Chapter 6, senders can specify a temporal constraint on messages, thus not requiring senders and recipients to be connected to the system at the same time.

Additionally, the dissemination mechanism is transparent to application developers and users: An application can send and receive messages simply by connecting to the system, sending messages and registering contexts, respectively. The activities that lead to messages being delivered are of no concern to developers or users.

In the next section, we discuss various requirements for a large-scale context-aware communication system. They serve to better understand the challenges one faces when designing a framework such as Contextcast. They also reflect the guiding principles in the design of Contextcast, which we introduce in the remaining chapters.

2.2 Requirements for Contextcast

On the basis of the previous examples and discussion, we can now introduce the basic building blocks and requirements for our Contextcast system. Obviously, an integral part of Contextcast is a formal specification of the communication paradigm. This includes (1) a specification of client contexts, which are used as recipients for contextual messages, (2) a definition of the messages in such a system and, in particular, the way contextual messages are addressed, and (3) the matching semantics between the two, i.e., which message needs to be delivered to which client. All these form the basis of a context-aware communication system and we formally define them in Chapter 3.

However, while the semantics pertain to the effectiveness of a Contextcast system, the following requirements focus on the efficiency and scalability of such a system. We present solutions to these points in the later chapters, as laid out in Section 1.4.

1. *Efficient Contextcast Message Forwarding* aims to limit the forwarding of messages towards actual recipients. For reasons of scalability, Contextcast uses a

network of nodes for message dissemination. Broadcasting messages in such a network would ensure that all clients receive a message, in particular those whose context matches the addressing. It would, however, severely restrict the system's scalability. Messages should therefore only be forwarded to nodes that have matching recipients connected or are on the path to a node with matching recipients. To this end, nodes require information about connected clients and their context(s).

2. *Efficient Update Propagation* complements the previous routing approach. Nodes require client context information for efficient forwarding decisions. Broadcasting all context updates limits the system's scalability similarly to broadcasting messages. Therefore, in the interest of scalability, it is necessary to keep the amount of propagated context updates low. To this effect, the approach needs to reduce redundant information in context updates. Also, it should only propagate context updates to nodes that need it for their routing decision, i.e., towards senders of messages that a context might match later.

3. *Support for Temporal Addressing* extends the notion of client contexts to a temporal component. It allows to address client contexts that occur at some point in the past or future. This also augments the system with a comprehensive temporal decoupling. Temporal Contextcast requires an extension of the message semantics to include temporal constraints. Particular care must also be taken to ensure that the privacy of clients is not compromised, especially with regard to their archived historical contexts.

2.3 Classification of Related Work

In its essence, Contextcast is a group communication mechanism. It allows senders to address and disseminate messages to a group of recipients, specified via their context. This makes it similar to other communication systems that also offer $m : n$ communication patterns. There are, however, a number of aspects that set Contextcast apart from similar technologies. In this section, we describe these criteria and subsequently use them to classify related work.

2.3.1 Classification Criteria

Implicit Addressing

Contextcast uses *implicit addressing* to specify the recipients of a message. Implicit addressing means that recipients are not specified in terms of their (explicit) network

addresses. Instead, Contextcast enables senders to define the set of recipients using constraints on their context attributes. Since these constraints can be rather complex, this allows for a very fine-grained specification of the recipients of a message. *Explicit addressing*, in contrast, uses the network addresses or an address of a recipient group to explicitly specify recipients, usually independent of their context attributes.

There are a number of advantages of using implicit over explicit addressing: First, implicit addressing decouples senders and recipients. A sender does not require detailed information about a particular recipient, which may be used to profile clients. Instead, they simply specify the context constraints and the Contextcast system takes care of the message dissemination. Nor do recipients need to know about possible messages or senders, they merely register with the system to receive messages addressed to them. Second, implicit addressing provides very fine-grained recipient groups. An addressing can be as specific as to address only a single client or as broad as to address most of the clients on a continent. To support such a fine granularity for explicit addressing would either require listing every recipient in every message or creating an enormous amount of recipient groups to address (basically one group for each element of the power set of the set of all clients). This poses a serious scalability limitation for explicit addressing schemes. Additionally, if a system with explicit addressing uses group addresses, these groups need to be established before addressing them, so clients can actually join the groups.

One disadvantage of implicit addressing, though, is an added overhead during message dissemination. Contextcast routers cannot simply lookup a single network or group address when making their forwarding decision, they need to evaluate the context constraints. This requires that the system knows about clients and their contexts, i.e., client context updates need to be propagated to the routers. This puts a significant load on the routing infrastructure, especially with dynamic client contexts. However, in our opinion, the added flexibility far outweighs this drawback. In this dissertation, we also focus on algorithms to cope with this update load that is generated by context updates.

Sender-centric Dissemination

Contextcast provides a sender-centric communication model. This means that senders specify the intended recipients of their messages. Such a recipient specification can, e.g., be a group address in explicit addressing or constraints on context attributes in implicit addressing. Such a sender-centric model requires little or no action on behalf of the recipients, thus allowing users to focus their attention on more important tasks.

The opposite is recipient-centric dissemination, where recipients specify what kind of messages they are interested in. This usually happens by clients joining a

multicast group or subscribing to a certain subject. Such a scheme, however, requires (1) that clients know about possible messages, and (2) subscribe to the ones they are interested in. The advantage, though, is that clients receive exactly the messages they have subscribed to.

A sender-centric communication frees users from this responsibility and allows senders to specify the intended recipients. And if a message is mistakenly delivered to a recipient, the receiving process can simply drop it, without informing the user—but possibly after sending feedback to prevent such messages in the future.

Complex Addressing Support

Contextcast provides a powerful scheme for addressing recipients. First, a message can specify an arbitrary set of *constraints* on *context attributes*. Only clients whose context matches these constraints receive the message. Other systems' support for such context information is either very limited or missing completely. Geocast, e.g., can address clients at specific locations, but does not provide any other context information for addressing.

In addition to this flexible context addressing, Contextcast also provides *temporal predicates* to further enhance its addressing power. Using these temporal predicates, which we discuss in detail in Chapter 6, senders can augment the addressing with temporal constraints. Thus, such messages address only clients whose contexts at some time in the past or the future match the addressing. Again, most other systems lack this ability when addressing messages.

2.3.2 Related Work

Our approach for a context-based communication system and the corresponding algorithms and protocols are related to a number of works in different fields. First, there are the multicast or group communication approaches, in particular on the network and application layer. The latter ones use an overlay approach on top of an IP infrastructure and are therefore similar to the overlay approach we present for Contextcast. Second, there are the various geographic routing methods, which include geocast, the management of spatial information or Peer-to-Peer (P2P) approaches with a coordinate structure. These techniques enable routing according to location, which is one part of a client's context, but fall short when it comes to addressing other context attributes. Third, there are various Pub/sub approaches, in particular content-based systems, which enable clients to select events according to constraints on arbitrary event attributes. This is closest to the context constraints we present for Contextcast, however, it requires clients to select events, whereas in Contextcast senders use these constraints to select recipients for their messages.

We present each of these areas in detail and classify it according to the criteria we introduced in the previous section. This section is intended as a general overview of related work. In later chapters, where we present additional optimizations for our system, we also present additional related work for each particular approach.

Multicast Communication

Multicast approaches in their broadest sense allow for one or more senders to address messages to a group of recipients, usually a subset of all hosts in a network [CD85]. They are also sometimes referred to as group communication mechanisms. Messages are sent to a *group address* or identifier and recipients are associated with one or more of these groups [Pow96]. Such a communication can take place on the Data Link Layer (for Local Area Networks (LANs)), the Network Layer, or the Application Layer. Since we consider an Internet-wide communication scheme, we focus our discussion on the latter two.

Network Layer. The famous "End-To-End Arguments" [SRC84] are that functionality should be added to higher layers in a network unless an implementation at a lower level greatly improves performance. The authors of [DC90] reason that for this performance benefit multicast should be implemented in the Network Layer. Various other authors, e.g., [BFC93, Moy94, DEF+96], have since researched efficient multicast dissemination in the Network Layer.

They all employ a tree-structure to efficiently disseminate messages to the group members. There are, however, two different basic types of such dissemination trees. *Source-based tree* approaches, e.g., [Moy94], maintain a separate dissemination tree for each sender. Such a source-based tree can be constructed to be optimal for each sender and a given group. *Shared tree* approaches, such as [BFC93], in contrast, use a single tree for each group. These shared trees are usually rooted at a (central) rendezvous node, towards which both messages and group joins are forwarded.

While this is usually does not lead to optimal trees for every sender, maintaining a single shared tree per group is less costly than maintaining one for every pair of sender and group. Because of the less costly maintenance, Contextcast—as we introduce in Section 2.4—also employs an undirected, acyclic graph as its shared dissemination structure.

Application Layer. While the previously mentioned works focus on multicast protocols at or below the network layer, for various reasons their adoption in existing infrastructures proved rather slow [CRSZ02]. For this reason, a number of alternatives to IP multicast were researched. (The authors of [ESRM03] provide an overview of these alternative approaches.) One particular class of alternatives are Application Layer Multicast / Application Level Multicast (ALM) approaches (sometimes also referred to as Overlay Multicast), which do not rely on modifications

to the existing infrastructure. Instead, their operation is *overlaid* on top of existing IP infrastructure. End systems form the dissemination tree(s) and handle the forwarding of multicast messages. Numerous authors, e.g., [PSVW01, RHKS01, BBK02, CRSZ02, Cha03], have explored such approaches.

There are different methods how these approaches establish the dissemination tree [ESRM03]. On the one hand, *mesh first* approaches begin by connecting all participants in a meshed network. Then a subset of nodes and connections is selected to establish a particular dissemination tree. Examples include Narada [CRSZ02] or Scattercast [Cha03]. *Tree first* approaches, on the other hand, directly connect a node to a parent node in the dissemination tree, based on metrics such as latency or available bandwidth. Yoid [Fra00] or Overcast [JGJ+00] belong to this group. (Also see [HASG07] for a much more comprehensive overview of different ALM approaches.)

Since modifying the network layer proved so difficult and slow, our Contextcast approach also focuses on an Application Layer implementation: Our dissemination network, which we introduce in Section 2.4, consists of end systems, operated by ISPs, NPOs, cell phone providers, etc. Over time, newer network components may offer this functionality as well and thus augment or replace the overlay network.

All such multicast approaches allow an efficient dissemination of messages to a group of recipients. However, they all use some form of previously defined groups for communication. Building a Contextcast system on this basis would require a mapping of client contexts to multicast group addresses. It would also need some type of scalable lookup service for this mapping. Even if such a service was available or could be constructed, it would hardly provide the fine-grained granularity that Contextcast aims to achieve: A concrete implementation might restrict the number of such groups. Internet Protocol Version 4 (IPv4) multicast addresses, e.g., are in the range from 224.0.0.0 to 239.255.255.255, or, to put it differently, use 28 bits. Thus, the system would need to map each client to one or more of these $2^{28} = 268\,435\,456$ groups, which match their context. But even if this number was much higher (as it is for Internet Protocol Version 6 (IPv6)) or even unlimited (with arbitrary group identifiers), it still would not be feasible: Let R be the set of all possible recipients, with $|R| = n$. To achieve the same fine grained addressing as Contextcast would mean maintaining $2^n - 2$ groups (one for each element of $\mathcal{P}(R) \setminus \{\emptyset, R\}$). This is obviously not scalable.

Explicit Multi-unicast (Xcast) [BFI+07] does away with multicast groups and the signaling and management overhead. Instead, each recipient is listed in a message. Messages are forwarded along an ad-hoc tree, with a router looking up the neighbor via which to reach each recipient. Whenever a message is duplicated for two or more links, the router splits the recipient set according to which recipient it can reach via each of the links. Such an explicit enumeration of recipients works well

for small groups (cf. also Small Group Multicasting [BFM00]). It does not scale well for a large-scale context-aware communication system such as Context. Also, this approach would still require a scalable directory service to lookup all the matching recipients. As we mentioned before, this in itself is not a simple task.

Summary. All these multicast protocols use explicit addressing and combine sender-centric dissemination (the sender addresses a given group) with a recipient-centric one (clients need to join a group). They offer no support for recipient context or a temporal addressing.

Geographical Addressing and Routing

Similar to the multicast approaches we discussed before, geographical addressing and routing also allows the addressing of groups of recipients and the corresponding dissemination of messages. In contrast to multicast, though, the senders specify recipients not by a group address or identifier but by their current location instead. We distinguish three main areas: geographical messages (commonly simply called "geocast"), distributed management of spatial information, and P2P Systems structured using an underlying coordinate system.

Geocast. Some of the earlier works in this area were presented in [Fin87] based on Cartesian addresses (or indeed any form of coordinates for which a metric distance can be calculated) and a corresponding routing approach. Another group pursued the idea further and presents various methods for geographical routing: (1) GeoRIP, GeoOSPF, and GeoBP extend IP routing mechanisms to allow for a geographical routing, (2) a geographical multicast approach, which maps locations to multicast groups, (3) a directory approach, where recipients (or rather, their subnets) are looked up in the Domain Name System (DNS) system, and (4) an overlay approach, which uses end systems for the dissemination of geographical messages. GeoRIP and GeoOSPF [Nav01] extend the Routing Information Protocol (RIP) [Mal98] and Open Shortest Path First (OSPF) [Moy98] protocols with location information for geographical routing, respectively. GeoBP [Nav01] uses a "broadcast and prune" approach, which is most similar to Distance Vector Multicast Routing Protocol (DVMRP) [WPD88]. All three rely heavily on the underlying IP infrastructure and its routing protocols. Contextcast, in contrast, should not depend on any underlying routing infrastructure since this limits the flexibility if the underlying structures change in the future.

The geographical multicast approach [IN96] maps locations to one or more multicast groups. The authors propose Protocol Independent Multicast Sparse Mode (PIM-SM) [DEF+96] for the actual dissemination of messages, which floods the information about each group throughout the network. This, combined with the large amount of groups necessary for fine-grained geocast leads to a large

amount of management traffic. The authors also address this point and propose a hierarchical approach, where only the group information for larger areas are propagated network-wide. Hierarchically coded addresses nevertheless ensure that all groups can be reached from every node. While this approach suits the containment relation of the location attribute, it does not handle a context-aware addressing very well. The recipients can have arbitrary attributes and values, often with no useful containment relation that could be exploited. The necessary number of groups for a fine-grained addressing, as we have already argued for multicast approaches, would simply not be scalable.

The directory-based approach [IN99] uses DNS to look up all the access nodes whose service area intersects the target location. A message is then sent to all those access nodes using regular unicast mechanisms. There are some performance and scalability problems for such an approach: DNS with its structure following administrative domains instead of geographical ones is rather inefficient for such a lookup. Even if we assume that this problem can be solved, the sending of several (identical) unicast messages places a substantial load on the network. And as with the geographical multicast approach there is still the open question how Contextcast with its attributes other than location could use such a system.

The overlay network approach [NI97] achieves geographical routing on top of an existing IP infrastructure. The overlay network consists of *GeoRouters*, which forward geographical messages. Access networks contain *GeoNodes*, which connect clients in their service area to the overlay network. A client wishing to participate runs a local service, the *GeoHost*, which acts as a local endpoint for applications using the geocast service. The *GeoRouter* network is structured hierarchically, with routers higher up in the hierarchy covering the service areas of all routers below them. A router forwards a message to all its child routers whose service area intersects with the target of the geocast message until it reaches the *GeoNodes*. Also, a message is forwarded to a node's parent if the target area is not completely contained in the node's service area. Such a hierarchical design works well for the geographical containment with location, however, it does not easily allow other, arbitrary context attributes.

The Nimbus framework, described in detail in [Rot05], provides a platform for various location-aware services. Among others, it also offers a semantic geocast [Rot03] service to applications. It differs from the previous approaches mainly in the fact that locations are not addressed using geographical coordinates but using semantic (or symbolic) locations instead. It is built on top of a Location Server Infrastructure (LSI), which manages mobile clients' current position. Since the LSI manages the positions of mobile clients, it could also be categorized as a system for the management of spatial information. However, as the geocast implementation relies heavily on the LSI's architecture, we choose to present it here. The Location Servers are intercon-

nected in a tree topology, mirroring the containment relation of locations. Nimbus' LSI allows several such hierarchies, reflecting different organization schemes, e.g., continents, countries, cities, etc. as opposed to an organization via geographical features such as mountains, rivers, etc. In addition to the usual parent-child relation in a hierarchy, (partially) overlapping locations between different hierarchies can have "associations". This enables, e.g., an association of the geographical feature of a particular mountain with the state it is in; thus, a router can forward a message addressed to the mountain also to the router of the associated state, reaching clients that are connected to an associated location in a different hierarchy. It also supports shortcut links for a more efficient forwarding of geocast messages. However, Nimbus is also limited to locations, it does not support arbitrary context attributes.

The geocast system presented in [Dür10] also offers geographical messages. It supports symbolic, geometric, and hybrid addresses, through an advanced location model [DR03]. The architecture of the system resembles the GEO system by Navas and Imielinski: The actual message dissemination is handled by an overlay network of Geocast Routers, with Geocast Message Servers providing access to clients in a certain area [DBR05, DBR06]. The Geocast Routers are connected in a tree that approximates the lattice structure of the location model. Forwarding in this network takes place in two phases: First, a given message is routed to a router in the target area. Second, the message is distributed to all Geocast Routers and Message Servers in the target area. For the first phase, there generally exist multiple paths between a router and the target area in a lattice model. Thus, the first phase calculates a least ascending path between the sender and recipient. This reduces the load on the routers higher in the lattice, which would become the bottleneck if the system were to route every message, e.g., via the *world* router. It then searches for a known router on this path, choosing the one closest to the target area if several exist, and forwards the message. In the second phase, the Geocast Routers use flooding to disseminate the message to all routers whose service area is completely within the target area. To further lower the load on routers higher up in the hierarchy, the authors enhance the overlay network by additional dynamic shortcut links [DR08]. Routers maintain these links adaptively to some distant locations that are addressed often. When such a shortcut exists, a message can then be forwarded directly, without ascending in the tree. As a geocast system, however, the approach supports (symbolic, geometric, and hybrid) location addresses only, no additional context attributes.

Generally, all the geocast systems we have discussed in this section provide an addressing for a recipient's location attribute only. In contrast to our context-aware communication, they do not consider additional context attributes. Neither can these additional attributes be added to any of the systems in a simple manner since they usually rely on the hierarchical nature of locations to improve the efficiency of their approach.

Distributed Management of Spatial Information. The focus of such systems is the distributed storage and retrieval of information with a relationship to a spatial location. First, there are the Location Service [Leo03] and the Source Description Class (SDC) Tree [LDR10, Lan10], which were developed in the Nexus research project. (For an overview of Nexus, see Section 1.2.3.) The Location Service uses a spatial partitioning to manage mobile objects. The objects are stored in Location Servers; the selection of concrete servers uses the position of the object and the server's service area. To allow for efficient query resolution, the servers are interconnected in a hierarchical fashion, modeling the spatial containment of locations. When a server receives a query, it checks which of its children intersect the query area and forwards it to those children. If its own service area does not fully cover the query area, it also forwards it to its father in the hierarchy (which covers a larger service area). This forwarding ensures that eventually all Location Servers whose service area intersects the query area receive the query. The SDC Tree and its accompanying description formalism extend this idea. Instead of limiting the description of a server's information to location, they allow a description using several *defined classes*. These defined classes describe information by a base class and zero or more relations and constraints, one of which may be *location*. Each node in the SDC Tree has a node class associated. The tree itself is structured using the subsumption relationship of defined classes, with each node class subsuming the node classes of its children. While the defined classes with its attribute constraints is somewhat similar to a client context in Contextcast, its design is very specific for describing the information a source can provide. Usually, this is very narrow, e.g., a server might provide floor plans for the Museum of Natural History but probably will not contain a temperature map of downtown Manhattan. Client contexts in close proximity, in contrast, can be rather diverse. Also, such descriptions are by their nature mostly static, in contrast to client contexts, which can change often.

RectNet [Heu05] is a similar system for the management of spatial objects. It partitions the space in rectangular areas, which are managed by a single server or cluster head. Partitions are split dynamically when the load of a cluster head gets too high. Two new cluster heads then manage the two new partitions, while the original cluster head becomes their parent in a tree, forwarding queries to its children. This leads to a similar hierarchical structure as for the Nexus Location Service, with inner nodes forwarding queries along the tree to the leaf nodes. In addition, it offers a flat routing mechanism, with nodes connecting to their spatial neighbors. This is similar to the Content Addressable Network (CAN) P2P system, which we are discussing later in this section. The focus of RectNet is clearly a spatial message distribution, similar to the geocast methods we have discussed previously. As such, it does not use additional context attributes for message addressing and dissemination.

The same author has presented another system called ContextCast [Heu02], whose name is similar to the system we introduce here. Its design is very similar to RectNet: It uses a global context space $G \subset \mathbb{Z}^n, n \in \mathbb{N}$. A context space S is is a subcube S^* of this global context space G with a unique name. Cluster heads manage the individual context spaces. Whenever a cluster head is in danger of becoming overloaded, its space is split into 2^n subspaces, each with a newly appointed cluster head, which become the children of the original cluster head. The routing uses the resulting tree to reach all context spaces that intersect a query region; in contrast to RectNet, there is no flat routing scheme. While the author focuses on a location service, a context space can contain more dimensions to represent additional context properties of a client. At first glance, this could be used to place client contexts in such a context space and then address cubes in this space. However, it is unclear how $G \subset \mathbb{Z}^n$ would support arbitrary context attributes that do not map easily into $\mathbb{Z}$, such as a type hierarchy or strings. Another difficulty is the placement of client contexts which specify only some of these n attributes. Where does the system place such a context in the missing dimensions?

P2P Systems with Coordinate Structure. A Content Addressable Network (CAN) is an indexing scheme for information in P2P system [RFH+01]. Its design is similar to the ContextCast or RectNet systems by Heutelbeck we described before: CAN uses a virtual d-dimensional Cartesian coordinate space $[0,1]^d$, which—for the purpose of message forwarding—forms a d-torus. This coordinate system is partitioned among the nodes that form the system, with each node responsible for its partition. A CAN is able to store arbitrary (key, value) pairs. To this end, a key is mapped to a point P in the d-dimensional coordinate system using uniform hash functions. The pair (key, value) is then stored on the node that is responsible for the partition that contains P. Storing and retrieving a pair relies on routing in the CAN as follows: Every node maintains a list of its neighboring zones and the IP address of the node responsible for each zone. A store or retrieve request to the coordinates of the destination point P_d is greedily forwarded to a neighboring zone (and thus the node responsible for it) that is closer to P_d than the current node. Due to the d-dimensional structure, it is very easy to map, e.g., 2-dimensional GPS coordinates into such a space and thus use an object's physical coordinates for routing. Also, just as for Heutelbeck's ContextCast, one could propose mapping other context attributes into additional dimensions of this d-dimensional space. However, this would lead to the same difficulties with the mapping of certain types and the absence of attributes that we described before.

Globase.KOM [KLS07] is a P2P system, which manages the position of the participating peers. Each peer has a physical GPS coordinate, which is directly represented in the system. Globase.KOM uses a superpeer structure, i.e., only a subset of

nodes—the superpeers—handle queries. The system dynamically detects areas of high load in a zone and clusters the nodes in this area into their own child zone and appoints a new superpeer for each zone. A superpeer maintains connections to its parent superpeer and to all its child superpeers. This leads to the familiar tree structure, with child nodes contained in the zone of their parent. A regular peer connects to the superpeer that manages the region it is located in. Routing a message in this structure also follows the usual pattern: To lookup a given location, a node queries the superpeer it is associated with. Each superpeer then checks if the queried location is inside its own zone. If it is, the superpeer also checks if it is inside the zone of one of its children. It then forwards the request either to the child superpeer or directly to the responsible node. If not, it forwards the request to its parent superpeer (which covers a larger area; if necessary, this is repeated until the root, which covers the whole area). In addition to this lookup operation, the system also supports range queries and nearest-neighbor queries. Again, Globase.KOM does not provide any means to address peers other than their location. As we have argued before, it is very difficult incorporating context attributes into such a tree structure that is explicitly based on physical location and the containment relationship of areas.

Summary. All these geographical addressing and routing approaches offer implicit addressing and a sender-centric dissemination. They do not, however, support more complex addressing involving context attributes other than location or any temporal relations.

Publish/subscribe: Topic-based and Type-based

Due to the large nature of the Internet, distributed systems have also grown more complex and more dynamic over the years. This required a more flexible communication paradigm than synchronized point-to-point connections between hosts. Pub/sub or event-based systems [BBMS98] provides such a looser interaction scheme between nodes. Recipients (or subscribers) can express their interest in certain events and are notified by the system when a sender (usually called publisher) generates a matching event. An event manager as middleware is responsible for notifying all subscribers about the events matching their subscriptions.

The main strengths of event-based systems are a *time*, *space*, and *synchronization* decoupling of senders and recipients [EFGK03]:

- *Space decoupling:* Publishers do not need to know about Subscribers and vice versa. An event service is responsible to match generated events to subscriptions and ensure their delivery.

- *Time decoupling:* It is not necessary for the publishers and subscribers to interact at the same time. In particular, a subscriber might receive an event when the publisher is no longer connected to the system.
- *Synchronization decoupling:* Both producing events and notification about events happen asynchronously. That is, neither publishers nor subscribers are blocked by the interaction.

This decoupled interaction scheme of Pub/sub systems make them well suited for use in mobile environments with intermittent connectivity [HGM04]. Contextcast provides a very similar communication scheme with the same decoupling properties.

The authors of [EFGK03] distinguish three different types of Pub/sub systems, depending on the form that subscriptions are specified: Topic-based, Type-based, and Content-based. We discuss the first two approaches in this section. Content-based Pub/sub is more similar to Contextcast, thus we discuss it in detail in its own section.

Topic-based Publish/subscribe. The first Pub/sub systems were based on *topics* (or *subjects*). Senders publish events, specifying their topic by keywords, and recipients can subscribe to these topics. Examples of such systems include iBus [AEM99] or TIBCO Rendezvous [TIB08].

Topics are very similar to groups used in group communication, thus the challenges and problems are very similar as well. Subscriptions to a particular topic are equivalent to becoming a member of a group for this topic; and publishing an event with a given topic is the same as sending a message to a group for this topic. In addition, most topic-based Pub/sub systems extend the flat addressing of distinct groups using a hierarchy of topics [OPSS93, EFGK03]. This allows to arrange topics using a containment relationship. Subscribing to a particular topic in the hierarchy then also subscribes to its subtopics. But even with this extension, it is still very difficult to accurately replicate a client's context with subscriptions to a limited number of topics and offer the same fine-grained addressing that Contextcast aims for.

To summarize, topic-based Pub/sub uses explicit (keyword) addressing, a dissemination that is both sender-centric and recipient-centric, but provides no additional addressing concepts using context attributes or temporal constraints.

Type-based Publish/subscribe. Events for a common topic often exhibit structural similarities in addition to the similarities in content. Therefore, type-based Pub/sub systems classify events according to their type, with subscriptions specifying interest in certain types, e.g., [EGD01]. These types can then be represented by classes and objects, allowing for a closer integration of the language and the middleware. In particular, it enables static type checking at compile time.

Another example of such a system is Hermes [PB02, Pie04], which the authors call a type- and attribute-based Pub/sub system. In their model, every event has an associated type, which ties it to a programming language's type. Each type has a set of attributes, i.e., the contents of the event. Despite this, the strong connection with programming language types limits the use of attributes to the respective event type. The model is therefore less flexible than content-based Pub/sub (discussed in the next section), which can mix arbitrary attributes in a single event.

With respect to our classification, type-based Pub/sub is very similar to topic-based systems: Addressing uses explicit (type) information, the dissemination is both sender- and recipient-centric, but again it offers no support for context addressing or temporal constraints.

Summary. Both the topic-based and the type-based Pub/sub systems offer an explicit addressing via the event topic or its type, mixed sender-centric addressing, but no support for contextual or temporal addressing.

Publish/subscribe: Content-based

The expressiveness of topic-based and type-based Pub/sub, despite extensions such as hierarchical topics or support for wildcard subscriptions, is limited. Content-based Pub/sub, described, e.g., by [CRW00], in contrast, overcomes this limitation by a subscription scheme that directly uses an event's actual *content*, not some topic or type.

Most content-based Pub/sub systems use a common definition of events and subscriptions, which we briefly introduce here [SA97, Car98, CRW00, FJL+01, Müh01, FJLM05]. Informally, an event is a set of attributes, each of which is a (name, value) pair. A subscription contains a set of constraints on these attributes, usually in the form (name, operator, value), describing matching events. Some systems, such as Siena, also explicitly include the attribute type in the definition, thus an attribute becomes (type, name, value) and constraints become (type, name, operator, value) [Car98, CRW00]. Multiple constraints in a subscription are usually combined using an implicit logical AND (i.e., a given event must match all constraints) but could also use explicit logical operations for more complex subscriptions (such as in Elvin [SA97]). We use a scheme very similar to these proven and wide-spread definitions for our Contextcast system, which we introduce in Chapter 3.

Due to scalability considerations, content-based Pub/sub systems are usually built as distributed systems. A network of brokers handle event dissemination to all clients with a matching subscription. This dissemination can employ a variety of different routing strategies, described in [Müh02]: The simplest strategy is a *flooding* of events with a local filtering performed by the subscribers. Broadcasting events, however, limits the scalability of the system, especially if each subscriber is

only interested in a small percentage of events. Thus, a number of more advanced *content-based routing* strategies exist that evaluate subscriptions to limit dissemination of messages. In general, these filter-based routing approaches forward subscription information between brokers, which use them for their forwarding decisions.

The approaches differ in the way they handle redundancies between different subscriptions, thus reducing the amount of information that is maintained by the brokers:

- The *Simple Filter-Based Routing* floods each new and canceled subscription into the broker network. Each broker therefore knows all subscriptions and can forward events to neighboring brokers accordingly.
- *Routing based on Filter Identity* ("identity-based routing") removes identical subscriptions from the routing process: A subscription F is not sent to a neighbor if another identical subscription G was forwarded to that neighbor.
- *Routing based on Filter Covering* ("cover-based routing") extends this to subscriptions to subset of events: A subscription F is not forwarded to a neighbor if another subscription G was already forwarded that matches a superset of events of F.
- *Routing based on Filter Merging* ("merge-based routing") extends the idea of filter covering further. Merging does not solely rely on client subscriptions but can generate subscriptions that are a superset of client subscriptions: A set of (client) subscriptions $\{F_1, \ldots, F_N\}$ is not forwarded to a neighbor if a broker forwards a subscription G instead, such that the events matched by G are a superset of all the events matched by $F_1, \ldots, F_N$.

Additionally, the routing strategies can also include *advertisements* to further improve the handling of routing information [Car98]. Advertisements are descriptions of possible events, which are broadcast by producers. The brokers can then use this information to forward subscription information only towards interesting producers (using any of the previously described routing strategies).

Contextcast's context-based routing is based on these content-based routing strategies. Section 3.3.3 presents a reference context-based routing algorithm that is modeled after the simple filter-based routing. Chapter 4 shows an improved routing algorithm that is a generalization of the identity-based, cover-based, and merge-based strategies. In Chapter 5, we introduce a mechanism for adaptive routing information, which generalizes the idea of explicit advertisements.

Numerous content-based Pub/sub systems have been developed over time, such as Elvin, Gryphon, Siena, JEDI (Java Event-based Distributed Infrastructure), Rebeca,

and PADRES. They are similar in many ways, but vary in details due to different focuses of the various systems.

Elvin [SA97] originally started out as a centralized server, which delivered all events to all attached clients. Over time, it evolved into a server architecture that handles subscriptions and the dissemination of events according to client subscriptions. One focus was support for federation, thus servers can become clients of one another. Such a system can be scaled rather easily, by simply adding additional servers to handle client subscriptions and interconnecting these servers. However, the authors never detail how such a system disseminates events among the servers and to subscribers. Subscriptions in Elvin are complex boolean expressions on the attributes of an event. The available attributes are limited to integer, floating point, and string types, with associated comparison operators. Because of the flexibility of the event structure, the authors considered additional types unnecessary. Even though, it is possible to extend the system with additional data types.

Gryphon [BCM+99] utilizes a network of brokers to disseminate events to subscribers. Their focus is the efficient dissemination of events to all subscribers. To this effect, they describe a distributed version of the *link matching* algorithm (detailed in [ASS+99]): Instead of directly computing the subset of all subscribers that need to receive an event, each broker only computes to which of its neighbors it needs to forward a given event. To compute these neighbors efficiently, each broker maintains a Parallel Search Tree (PST) of client subscriptions. This PST is then searched for each event to determine the links over which the event needs to be forwarded. However, to compute this PST, each subscription needs to be broadcast. It is thus a system that uses the simple filter-based routing as discussed before.

Siena [CRW99, CRW01] also relies on a network of brokers to disseminate events. The authors introduce different topologies that can be used to connect the brokers: a hierarchical client/server architecture, an acyclic P2P architecture, or a general P2P architecture. These architectures differ in their tendency to overload individual nodes, their resilience to node failures, and the complexity of the management of the broker network. In addition, the authors describe two routing approaches: The first one broadcasts subscriptions and is thus identical to the previously discussed simple filter-based routing. The second one broadcasts advertisements and then only forwards subscriptions towards producers of matching events. In [CW03], they introduce the Siena Fast Forwarding (SFF) algorithm, which can efficiently disseminate events based on subscription information. SFF uses a counting approach, similar to [YGM94], to determine the set of subscriptions that a given event matches. In [KCW11], the authors extend Siena with support for 2D spatial objects, which can be used in both events and subscriptions. They also present an indexing approach to efficiently evaluate spatial relations between objects, which is necessary for an efficient forwarding of events in such a system.

JEDI [CNF01] builds on the notion of Active Objects (AOs), which interact with each other by sending and receiving events. A JEDI event consists of the event name, a set of strings, and further event parameters. An event dispatcher is responsible to deliver an event to all the clients that are interested in it. The authors present both a centralized and a distributed implementation of the event dispatcher. The distributed version uses a hierarchy of distribution servers, in which both subscriptions and events are routed towards the root as a kind of rendezvous node. This poses the same problem of overloading higher servers as in Siena's hierarchical architecture.

Rebeca [Müh02] uses a network of brokers connected in an acyclic topology for event dissemination. This simplifies the algorithms since nodes do not need to worry about duplicate events via additional paths. It makes the design vulnerable to failures of nodes, though. The author also presents and compares several alternative routing strategies, which we already discussed before. The authors of [FGKZ03] extend Rebeca with "location-dependent subscriptions", allowing consumers to incorporate their current location into subscriptions. The approach does not support client context beyond simple position information, though.

PADRES [FJLM05] employs an overlay network of brokers as well. It relies on the same scheme of publish-subscribe-advertise as Siena: Advertisements are broadcasts and subscriptions then forwarded towards those producers that may generate interesting events. Of particular interest in PADRES is the possibility to subscribe to past events in addition to the common future events in other Pub/sub systems [LCH+07]. To enable this, PADRES brokers store events in a database upon publication. Later, when a subscription to past events is issued, the brokers can re-publish the matching events from their database. We discuss this in more detail in Chapter 6, when presenting the temporal extensions to Contextcast.

Summary. Content-based Pub/sub provides implicit addressing via the event content and recipient-centric addressing using subscriptions. In its pure form, content-based Pub/sub does not allow specifying a recipient context. There are, however, some efforts to extend the content-based routing approach with context information. We discuss these approaches in the following section. One Pub/sub system, Rebeca, offers support for a small part of the client's context, the location. Another system, PADRES, provides a form of temporal addressing, allowing subscribers to access past events.

Publish/subscribe: Context-aware

Apart from the previously mentioned extension to Rebeca [FGKZ03], with its location-dependent subscriptions, plain content-based Pub/sub offers no support for recipient contexts when disseminating events.

The authors of [CdC05] present a similar extension, offering location-awareness in Pub/sub systems: Consumers register their position, in addition to any subscriptions, with the middleware. The location information is forwarded between the brokers, which maintain location tables in addition to subscription tables. The location table contains the locations of all of a broker's clients as well as all the locations it can reach via one of its neighbors. Producers can publish events to recipients in certain areas only. In the outlook of the paper, they also present a vision to generalize this approach to a *context-based routing* approach.

In [CM08], they sketch a system for "Context-aware Publish/subscribe", developed independently of our approach. They detail this approach further in [CMM09]. In particular, they introduce a fire monitoring system as an example, which benefits from context-aware Pub/sub. Based on this example, they argue a reversal of the matching semantics compared to content-based Pub/sub. Traditionally, events in content-based Pub/sub contain data in the form of (key, value) pairs. Subscriptions are the clients' constraints on these data, specifying interesting events. The authors show that their fire monitoring example requires the inclusion of a datum (the recipient's location) in a subscription. Also, publishers need to specify a constraint in events to select matching recipients. Therefore, in addition to regular events and subscriptions, their interface includes a `setContext()` function to inform the Pub/sub system about the context of a node. Subscriptions are augmented to also contain constraints on the context of the sending node. Events contain the usual data as (key, value) pairs as well as additional constraints on client context. Semantically, this leads to a system that is neither fully sender-centric nor fully recipient-centric. Both publishers and recipients can specify constraints on event data and contexts.

The authors also introduce their Shortest Path Context Forwarding (SPCF) routing algorithm. It relies on a link state protocol to learn the broker topology and calculates a shortest path tree rooted at each broker. Each node maintains two routing tables:

1. A *context table*, which contains for each broker the set of contexts of its clients.

2. A *content table*, which stores for each broker B_p each context c_p of the clients connected to B_p and each neighboring node N a list of content subscriptions and contexts from clients that are in a subtree below the neighbor N in the shortest path tree rooted at B_p (i.e., for which N is the next hop).

Message forwarding then becomes a two step process. The first broker looks up the client contexts that match the constraint of the event in the *context table* and encodes the resulting brokers in a Bloom filter [Blo70], which is appended to the event. Then, each node checks the relevant entries in the *content table* for this originating node, context, and each neighbor to find a matching subscription. The

Bloom filter from the first step is used to limit the search only to those subscriptions with a matching context.

Summary. This context-aware Pub/sub is similar to our Contextcast approach but also different in many ways. First, we argue a similar reversal of the semantics in Chapter 3. The Contextcast system, however, uses simpler, purely sender-centric semantics. Messages contain only constraints on client contexts and a single message payload, while clients only register their contexts, which are then used to deliver the messages. This frees users from the responsibility of explicitly selecting interesting messages. Second, Contextcast focuses on efficient routing algorithms, presented in Chapter 4 and Chapter 5, to reduce the amount of load placed on the system by the maintenance of client context information. This is especially important with regard to the dynamic nature of client contexts. Context-aware Pub/sub does not consider these issues. Third, Contextcast includes an extension for temporal addressing and dissemination of messages to clients with a certain context sometime in the past or future. This is not possible with context-aware Pub/sub. Additionally, temporal Contextcast focuses on questions of privacy combined with efficient message delivery for such messages. We present this extension in Chapter 6.

Summary

In this section, we presented an overview of various communication mechanisms similar to Contextcast. We used the criteria detailed in Section 2.3.1 to classify these different approaches. Table 2.1 shows the results of this classification at a glance, contrasting Contextcast to similar, related communication paradigms. No other approach offers the same combination of implicit addressing, sender-centric dissemination, and flexible addressing supporting both client context and a temporal addressing of clients.

2.4 System Model

In this section, we are introducing the Contextcast system model, which forms the basis for all the algorithms and improvements we are discussing in later chapters.

The Contextcast system consists primarily of three components, which form the Contextcast overlay network. These resemble Geocast overlays such as GEO [IN99] or the system of Dürr et al. [DBR06] in name and functionality. Similar concepts can also be found in various distributed content-based Pub/sub systems.

- *ContextHosts* are running on the client devices. They provide the interface between applications and the Contextcast system.

Related Work	Implicit Addr.	Sender-centric Dissemination	Complex Addressing	
			Contextual	Temporal
Multicast	No	Mixed	No	No
Geocast	Yes	Yes	Location	No
Topic-/Type-b. Pub/sub	No	Mixed	No	No
Content-based Pub/sub	Yes	No	No/Location	No/Past
Context-aware Pub/sub	Yes	Mixed	Yes	No
Contextcast	Yes	Yes	Yes	Yes

Table 2.1: Classification of Related Work

- *ContextNodes* are basically Contextcast routers but also provide access to the clients in their service areas.
- *ContextRouters* provide the actual dissemination of contextual messages from a sender to the addressed clients.

Figure 2.1 shows an overview of these components and the Contextcast system. Their functionality is detailed in the following sections.

2.4.1 ContextHost

A ContextHost is the local endpoint of the Contextcast system. Every client connecting to the network runs an instance of the ContextHost component. An application that wants receive contextual messages registers with the ContextNode. The ContextHost manages all these local application registrations and in turn connects to a ContextNode to receive contextual messages. Also, applications wanting to send such a message do so via their local ContextHost.

When a ContextHost receives a message from the network, it checks its addressing against the known context(s) before it delivers it to applications. This step is necessary since ContextHosts may receive messages that do not match their client context(s). This happens especially when the connection between a ContextHost and a ContextNode uses a broadcast medium, such as a WLAN connection. But even in the case of a wired connection, ContextNodes may deliver a message to all ContextHost using a broadcast in their subnet for efficiency reasons.

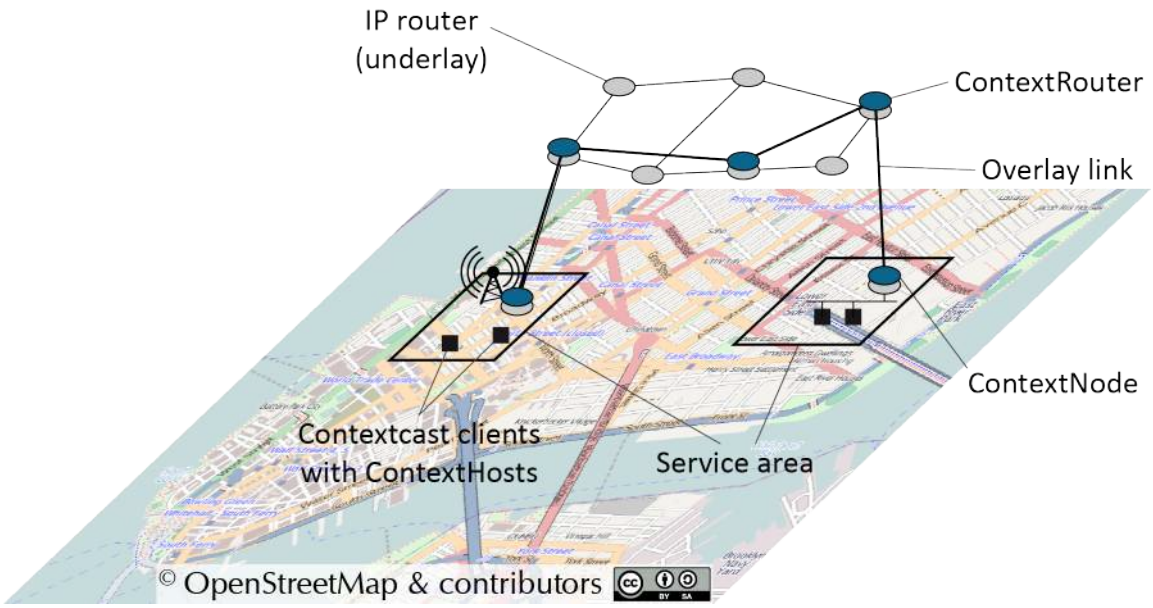


Figure 2.1: Schematic overview of the Contextcast system

Since its functionality and implementation are rather straightforward, we do not discuss the ContextHost component further in this dissertation. Instead, we focus on the efficient dissemination of messages in the overlay network.

2.4.2 ContextNode

A ContextNode is, in essence, a ContextRouter (see the following section) with the additional task of delivering messages to clients in an access network. For this purpose, each ContextNode has a certain service area assigned. This service area can simply be specified as a polygon of WGS84 coordinates or using a different model, e.g., on based on symbolic coordinates. All that is required is that it is possible to check a given position for containment within a service area. Clients within a ContextNode's service area connect to this ContextNode.

The concept of a service area results mainly from the use of wireless connections for mobile clients. However, for uniformity, we also extend this concept for wired connections to the ContextNodes. Additionally, we assume a certain local similarity of contexts, which is well represented in this design. Students on a campus, e.g., all have the type student, are all of a similar age, and for each of the different majors there are a number of students studying it. This similarity may not always be as pronounced but the concept of a service area in a wired network does not have any negative impact, either. Thus, ContextNodes that do not offer a wireless connection to clients have a service area assigned as well.

If the service areas of two or more ContextNodes overlap and a client is in the overlapping region, it can connect to any one of the ContextNodes. The ContextHost can base its selection on any number of factors, such as the bandwidth to each of the ContextNodes or the cost of data traffic for each link.

2.4.3 ContextRouter

The ContextRouters are responsible for the global dissemination of a message to all ContextNodes which have clients with matching contexts connected. The exact nature of the context-based routing information and the forwarding decision is part of a concrete context-based routing algorithm. We present a basic version of such a routing algorithm in Section 3.3.3. To avoid a broadcast of contextual messages, it uses information about contexts that can be reached via each link to achieve a directed forwarding only towards recipients matching the addressing of a particular message.

The management of such routing information can cause a considerable amount of traffic, though, especially with highly dynamic client contexts. Thus, Chapter 4 and Chapter 5 introduce optimizations to the basic context-based routing, which reduce this traffic and thus improve the scalability of the Contextcast system.

2.4.4 Overlay Network

Contextual message dissemination in Contextcast is performed by a distributed system of ContextRouters. History has shown that changes to the Internet infrastructure usually happen rather slowly, probably due to the cost involved. Examples are the slow adoption of IPv6 or support for IP layer multicast. Therefore, Contextcast uses an overlay network of ContextRouters, which can be deployed without changes to the actual underlying infrastructure. Future network layer routers may implement this functionality, though, and thus provide Contextcast as a core service.

The ContextHosts are running on end systems while the ContextNodes and ContextRouters are operated by ISPs, cell phone providers, individuals, etc. ContextRouters and ContextNodes utilize general-purpose hardware, so these nodes are more powerful with regard to storage capacity and processing power than IP routers. In the future, the functionality of ContextRouters and ContextNodes may become part of IP routers.

The Contextcast nodes are connected using virtual links in an overlay network. The overlay links deliver messages in any order and and with an unbounded delay δ. While individual nodes and links of the underlying network can fail, we assume that its routing mechanism eventually (typically rather quickly) provides alternative paths. Due to the strong decoupling of senders and recipients, which makes

end-to-end guarantees difficult, the goal of Contextcast is a best-effort message dissemination. If an application requires more than a best-effort service, Contextcast can be extended with a system of sequence numbers, acknowledgments and retransmissions.

While the underlying routing mechanism shields Contextcast from failures to a certain extent, failures of overlay links or nodes can still cause message loss or a partitioning of the Contextcast network. We therefore choose an approach very similar to the *mesh first* method for multicast dissemination trees as employed by Narada (see [CRSZ02]), which we discuss further in Section 2.3.2: each node maintains a set of overlay links (with an upper limit of k links), effectively creating a mesh of Contextcast nodes; this increases the system's robustness to such overlay failures.

A new node is incorporated into the mesh of Contextcast nodes in the following way: it establishes a (temporary) link to an existing node, e.g., via a well-known DNS entry, and then sends a Contextcast message to the ContextRouters requesting a connection. This message is addressed to an area surrounding the new node's current location. From the list of responding servers, the new node randomly selects at most k nodes that have not yet reached their connection limit and establishes a (permanent) connection with them. (If less than k nodes respond or have free links available, the new node can repeat its message with a larger target area, effectively using an expanding ring search to find suitable nodes to connect.) This method to establish the mesh links ensures that links are favored between geographically close neighbors, which allows the system to exploit local similarities between clients. Other, more advanced methods of selecting and establishing links would also be possible such as measuring the proximity of two nodes by the similarity of the locally connected recipients instead of the geographical distance.

On the basis of this mesh network, the nodes determine a minimum spanning tree as follows: they continuously exchange topology information using link-state advertisements (see, e.g., [MRR80]). With this information, all nodes can decide on a single node r as the root of a single shared spanning tree for the overlay network. This can be the node with the lowest unique identifier, the one with the most available resources, etc. With the complete topology information available, each node can compute the same minimal spanning tree rooted at node r. This spanning tree forms an acyclic graph, which somewhat simplifies our context-based routing algorithms: the nodes need not consider duplicate messages caused by cycles, etc. In Chapter 3, we are going to show, however, how the propagation of client contexts can be used to construct individual spanning trees per access network.

The nodes in the overlay exchange periodic heartbeat messages with their neighbors in the minimum spanning tree to detect failures. If an overlay node or link fails, the meshed topology remains connected, even though the acyclic graph may not.

Repeating the above algorithm for the updated meshed network then ensures that the nodes compute a new acyclic topology without the failed node. Once the new spanning tree is complete after a failure, the nodes adjacent to the failure point can resend context information to cause an update of the routing information in other nodes. Later, if or when a failed node recovers, it can then be reconnected to the mesh as a new node and integrated into the Contextcast spanning tree again.

Chapter 3

Contextcast Semantics

> Man's achievements rest upon the use of symbols...
>
> *(Alfred Korzybski)*

This chapter introduces the fundamentals for our Contextcast system: the basic building blocks, user contexts and messages, the addressing semantic, and the dissemination. Together, these form a basic context-aware communication mechanism and serve as the starting point for the various algorithms and improvements we introduce in the following chapters.

In Section 3.1, we formally define client contexts and contextual messages. Section 3.2 specifies how client contexts are matched with messages to determine the recipients of a message. After that, we discuss the dissemination semantics and introduce a reference dissemination algorithm in Section 3.3.

3.1 Contexts & Messages

Contexts and messages form the primitives of a the Contextcast system. Client contexts describe clients of the system as a set of context attributes while contextual messages transport information to a set of recipients, specified as constraints on the context attributes. The Contextcast system is responsible to deliver each message to the addressed recipients.

The definitions of client contexts and contextual messages, which we present in this section, are to some extent similar to the definitions of events and subscriptions in content-based Pub/sub [ASS+99, CRW00, Müh01]. As we have discussed in Section 2.3.2, however, the semantics of Contextcast allow for a more natural addressing of clients by the senders via addressing constraints in the messages. In contrast, recipients in a Pub/sub system need to select interesting notifications by a

WGS84: *location*	= 40.704989709835 N, 74.01199257043 W
Hierarchy: *transport*	= /vehicle/motorized/car
WGS84: *destination*	= 40.683087227208 N, 73.951669689687 W

Table 3.1: Example of a context C

WGS84: *location*	$\in$ <polygon Manhattan>
Hierarchy: *transport*	$\subseteq$ /vehicle/motorized
WGS84: *destination*	$\in$ <polygon Brooklyn>
Payload: *payload*	= <information ...>

Table 3.2: Example of a message M

subscription in advance. This semantic difference has an important impact on the definition in the following sections.

Picking up the example from Section 1.1, Table 3.1 and Table 3.2 show a client context and a message, respectively. The context describes a client somewhere in Manhattan's financial district, driving in a car, who is on the way to a destination in Brooklyn. The corresponding traffic control message addresses clients inside of a given polygon in downtown Manhattan, driving some kind of motor vehicle, and with a destination in a polygon in Brooklyn. (For better readability, we present both the context and the message slightly different from the formal definitions, which we use in the following sections; the difference is purely cosmetic, though.)

3.1.1 Client Contexts

The definition of client contexts has its roots in the Augmented World Model (AWM) of the Nexus project (also see Section 1.2.3). The AWM provides a global shared world model for context-aware applications. It employs an object-oriented data model: attributes are grouped together to objects to describe entities [GBH+05].

A client context (or *context* for short in later sections), as shown in the example in Table 3.1, is such a set of context attributes. It represents the current context of the entity it is attached to, i.e., usually a user of the system. For reasons of privacy, a client can register several contexts, each describing a different aspect.

Definition 3.1 (Context)**.** A context C is a *set of context attributes* α_i: $C = \{\alpha_1, \ldots, \alpha_k\}$. Each of these attributes α_i is a tuple consisting of the type, name, and value of the attribute: $\alpha_i = (\text{type}_{\alpha_i}, \text{name}_{\alpha_i}, \text{value}_{\alpha_i})$.

For the attribute *location* we have shown in Table 3.1, the formal notation is therefore $\alpha_1 = (\text{WGS84}, \text{location}, (40.704\,989\,709\,835\,\text{N}, 74.011\,992\,570\,43\,\text{W}))$.

Contextcast supports arbitrary attribute types, such as *WGS84* [Nat97] coordinates, simple numeric types *integer* or *float*, *string*s, or *structured attributes* based on a hierarchy of values (e.g., to specify a means of transport hierarchy as in Table 3.1). The actual value that an attribute can take depends on its type. For WGS84 coordinates, e.g., possible values are points defined by a pair of latitude and longitude or regions, specified as polygons by a sequence of coordinate points. Both of these values are used in the example. For numeric types, it can be a single value, a set, or a range of values. Other types restrict the allowed values accordingly as well. In this chapter, we assume that values in contexts are single values for each type, e.g., a point in WGS84 coordinates. In Chapter 4, we show how this can be extended to a set of values, an interval, or ranges, e.g., an area defined by pairs of latitude and longitude.

Contextcast can be extended to support additional types beyond the ones mentioned; all that is required for a new type are a specification of the allowed values and a set of *matching operators* that can be used when addressing messages. The latter requirement is discussed in more detail in Section 3.2.1.

In some algorithms in this dissertation, the names and types of attributes are important, whereas the actual values of these attributes is irrelevant. For instance, this is the case when determining the set of attributes that two contexts share (see, e.g., Section 4.4.3). We use the notation $\texttt{Attributes}(C)$ if we want to refer to the set of attributes that make up C without their particular value, i.e., only their type and name.

While we support arbitrary attribute names and types, such a system in practice requires a model of supported attributes for application developers to be useful. This includes attribute names, types, as well as their respective matching operators. Such a system can be provided, e.g., by the AWM of the Nexus project, it can be specified by a concrete Contextcast implementation, or some entirely different source.

Section 3.2.1 introduces a minimal attribute model that is used throughout this dissertation.

3.1.2 Contextual Messages

Recall from Table 3.2 that messages in the Contextcast system differ from notifications in Pub/sub systems in one important aspect: Notifications in Pub/sub contain all information implicitly in the attributes of the notification, it is up to the recipients to select interesting events via an appropriate subscription. In contrast, Contextcast messages specify the recipients of the message explicitly but this specification typically does not convey any additional information. As such, a Contextcast message consists of an addressing that determines the recipients of the message and the

actual information, which is simply an opaque payload for the purpose of message dissemination.

Definition 3.2 (Contextual Message)**.** A (contextual) message M consists of a contextual addressing (see Definition 3.3) and an (application-specific) payload.

The addressing determines the recipients of the message via a set of constraints that the recipient's attributes need to match to receive the message. The details of the matching semantics between a contextual message and client contexts are discussed afterwards in Section 3.2.

Definition 3.3 (Contextual addressing)**.** The addressing of a message M is a *set of constraints* ϕ_i: $\{\phi_1, \ldots, \phi_l\}$. Similar to context attributes, a single constraint ϕ_i is a tuple with the type and name of the constrained attribute, an operator, and a value against which the operator is evaluated: $\phi_i = (\text{type}_{\phi_i}, \text{name}_{\phi_i}, \text{operator}_{\phi_i}, \text{value}_{\phi_i})$.

The constraint on the *destination* from the example shown in Table 3.2 is thus represented as $\phi_3 = (\text{WGS84}, \text{destination}, \in, \text{<polygon Brooklyn>})$.

In essence, the statements from Definition 3.1 regarding type, name, value, and operators of an attribute apply accordingly. Also, like the previously mentioned $\texttt{Attributes}(C)$, $\texttt{Attributes}(M)$ refers to the set of all attributes that are constrained by M, without the actual operator and value that are used to constrain the attribute.

3.2 Matching

Building on the definitions of client contexts and messages in a Contextcast system, we can now discuss the matching semantics between the two. This determines whether a client with a given context is to receive a message. While the definitions are straight-forward, a distributed system poses more challenges for determining a message's recipient set, though; we therefore discuss it in Section 3.3.

Let M denote a message, whose addressing contains a number of constraints $\{\phi_1, \ldots, \phi_l\}$. For each one of these, it is possible to determine whether a given context C matches it:

Definition 3.4 (Single constraint matching)**.** A context C *matches* a constraint ϕ_i—denoted as $C \sqsubset \phi_i$—if and only if (1) $\exists \alpha_i \in C : \text{type}_{\alpha_i} = \text{type}_{\phi_i} \wedge \text{name}_{\alpha_i} = \text{name}_{\phi_i}$ *and* (2) $\text{value}_{\alpha_i}\ \text{operator}_{\phi_i}\ \text{value}_{\phi_i} = \texttt{true}$.

In other words, for a context C to match ϕ_i, C needs to contain the constrained attribute, determined by its type and name, and evaluating the matching operator yields `true` (we are discussing matching operators in more detail in the following

section). Note that the notation "$C \sqsubset \phi_i$" was chosen to reflect the fact that every constraint ϕ_i defines a subset of all possible contexts, i.e., those contexts matching ϕ_i.

Using Definition 3.4, we can now define the matching with the complete addressing of a Contextcast message M:

Definition 3.5 (Message matching)**.** A context C *matches* a message M with addressing $\{\phi_1, \ldots, \phi_l\}$—again denoted as $C \sqsubset M$ to convey the fact that a message determines a subset of client contexts—if and only if $\forall \phi_i \in \{\phi_1, \ldots, \phi_l\} : C \sqsubset \phi_i$.

Put differently, a context C matches a message M if and only if it matches all the addressing constraints. Thus, all constraints must be fulfilled for a context to match a message.

It is straight-forward to extend this addressing with a logical disjunction: A message can contain multiple sets of addressing constraints. For a context to match one such set, all the constraints in the set must be fulfilled (a logical conjunction). The sets themselves are connected by a logical disjunction, though. I.e., a context C matches a message M if it matches at least one of the addressing sets of M. In the evaluation of our concepts, we use only a single set of addressing constraints and send a message potentially multiple times instead of a single message with multiple addressing sets. In practice, this means that the results incur a slight overhead compared to the more powerful approach of messages with multiple addressing sets.

3.2.1 Operators

As mentioned in Section 3.1.1, the definition of matching operators can be part of the specification of a particular Contextcast implementation or it can reuse an existing type system, e.g., provided by the Nexus AWM. When defining attribute types, such as a type hierarchy representing a client's mode of *transport*, a designer must specify the possible operators and their evaluation. Obviously, the matching operators for common data types should reflect established usage to conform to a client's expected behavior.

While a comprehensive attribute model is beyond the scope of this dissertation and also not required for the presented concepts, for consistency we are providing a minimal model that is used in the remainder of this work. However, throughout the dissertation, we give hints regarding particular requirements when extending this minimal system.

***WGS84* Attributes (e.g., *location*)** The type WGS84 is used to specify points in the WGS84 coordinate system, i.e., GPS coordinates.

In our simple model, WGS84 attributes support only the operator $\in$ to test for containment: The operator $\in$ yields `true` if and only if the client's *location* coordinate is within a specified polygon.

In the example from Table 3.1 and Table 3.2, the constraint WGS84: *location* $\in$ <polygon Manhattan> yields `true` if the client's coordinate in somewhere within the polygon in Manhattan.

***Integer* and *Float* Attributes (e.g., *age* or *speed*)** Numerical types are used to describe numerical properties, either integer or float.

Such attributes support the usual number of comparison operations for ordered sets, i.e., $<$, $\leq$, $=$, $\geq$, and $>$, with their usual evaluation.

***Hierarchy* Attributes (e.g., *transport*)** The type Hierarchy is used for attributes whose values represent some kind of hierarchical relationship. This could, e.g., be a type-like hierarchical relationship such as a client's mode of *transport*. In such a hierarchy, a parent node is more general than its children. For example, a node "car" could have the child nodes "sedan" or "convertible" and the siblings "bicycle" and "pedestrian". Obviously, the semantics of a specific attribute depend on the actual hierarchy that is associated with it.

In our model, Hierarchy attributes support the operators $=$ and $\subseteq$. The operator $=$ is used to test two nodes for equality, i.e., it evaluates to `true` if and only if the compared values represent the same node. The operator $\subseteq$ is used to test if a node is a descendant or equal to another node:

$$\text{Hierarchy } \textit{transport} : A \subseteq B = \begin{cases} \texttt{true} & \text{if } A \text{ is a descendant of or equal to } B \\ \texttt{false} & \text{else.} \end{cases}$$

The constraint Hierarchy: *transport* $\subseteq$ /vehicle/motorized in Table 3.2 would thus evaluate to `true` since "Hierarchy: *transport* = /vehicle/motorized/car" is a direct descendant of /vehicle/motorized. It would also evaluate to `true` for a hypothetical "*transport* = /vehicle/motorized/car/sedan", with a sedan being a more specialized type of a car.

***Temporal* Attributes (e.g., *time*)** Temporal attributes are used, e.g., to model the temporal validity of a context. This type and the attribute *time* are part of an extension to Contextcast to support a temporal addressing and dissemination of messages. They are introduced in more detail in Chapter 6.

3.3 Dissemination

In a distributed Contextcast system as described in Section 2.4, delivering a contextual message to the clients can be modeled as a two phase process: In the first phase, the Contextcast system needs to determine the set of access network with matching recipients connected. A copy of the message is forwarded to each of these access networks. Once a message has been forwarded to a ContextNode, in the second phase each one can deliver it locally to the matching clients in its service area. In this dissertation, we focus on the first part, the efficient dissemination of messages in a distributed system.

Since clients can register and deregister contexts any time, the set of ContextNodes with matching recipients depends on the time a message is sent as well as the time it takes to propagate the message throughout the network. In the following sections, we discuss such a dissemination of a contextual message in more detail; we introduce the notion of a *perfect dissemination* and then extend this definition to cope with the actual conditions in a distributed system, such as an unbounded link delay.

3.3.1 Perfect Dissemination

Let M_{t_0} be a contextual message sent at time t_0 and `LocalContexts`(N,t) refer to the set of contexts registered at node N at time t.

Definition 3.6 (Dissemination)**.** A *dissemination* forwards a message M to a *subset* of ContextNodes N_M, including the origin ContextNode N_o.

Definition 3.7 (Perfect Dissemination)**.** A dissemination is called a *perfect dissemination* if and only if the following condition holds for any message M_{t_0}:

$$\begin{aligned} N \in N_{M_{t_0}} \quad \Leftrightarrow \quad & (\exists C \in \texttt{LocalContexts}(N, t_0) : C \sqsubset M_{t_0} \\ & \vee (\exists N_{\text{dest}} \exists C \in \texttt{LocalContexts}(N_{\text{dest}}, t_0) : C \sqsubset M_{t_0} \wedge N \text{ is on} \\ & \quad \text{the path from } N_0 \text{ to } N_{\text{dest}} \text{ along the spanning tree})) \end{aligned} \tag{3.1}$$

In other words, a dissemination is called perfect if and only if a message is forwarded exactly to the subset of ContextNodes with (a) either at least one context C connected at time t_0 such that $C \sqsubset M_{t_0}$ or (b) on the path to such a node along the spanning tree. (Obviously, in addition to the nodes $N \in N_{M_{t_0}}$, the origin node N_o always possesses a copy of the message, even though there may not be a $C \sqsubset M_{t_0}$ present there.)

A perfect dissemination is the optimal scheme for disseminating a message to all recipients, therefore the name: Only those ContextNodes that have a matching context registered actually receive a message for local delivery to their clients;

additionally, nodes on the path to these nodes also receive the message for forwarding. Forwarding it to fewer ContextNodes means that some clients do not receive a message, forwarding it to additional ones without matching clients wastes bandwidth.

We call these messages that are falsely forwarded *false negatives* and *false positives*, respectively: A *false negative* is a message that is not forwarded, even though a matching recipient exists, while a *false positive* is one that is forwarded, despite the fact that there is no matching recipient. (The next section offers a more formal definition of the concepts of false positives and negatives in a distributed Contextcast system.)

In the absence of failures, a perfect dissemination is almost trivial in a system with a single, centralized ContextNode: Whenever a message is submitted to this node for dissemination, it has already reached the only and thus all ContextNodes in the system, which includes the ones with matching clients registered. It is, however, rather difficult to achieve in a distributed system with unbounded link delay, as we are discussing in the following section.

3.3.2 Dissemination in a Distributed System

Even without failures, determining the set of ContextNodes for a perfect dissemination of a message M_t is difficult in a distributed Contextcast system: Clients register their context with any ContextNode in the network, messages can originate from any node as well. To forward messages from any source to the ContextNodes with matching client contexts, there exist a spectrum of approaches in such a system, with the following two marking the extreme ends:

1. Without context knowledge available on the routers, a message is broadcast to all nodes in the network; this way, all nodes where matching contexts are registered receive the message. However, typically also nodes without matching clients and not on the path to a node with matching clients receive the broadcast message, thus violating the perfect dissemination and causing unnecessary network load in the form of false positives.

2. With context information available on the ContextRouters, they can forward messages to only the access networks with matching clients, therefore achieving a perfect dissemination. However, with an unbounded delay δ, a ContextRouter cannot conclude from its lack of knowledge of a matching recipient in an access network that indeed there is no such recipient; the information could simply be still being transmitted. While this could be overcome by storing messages until a matching recipient is known, this would require storing messages indefinitely.

Neither broadcasting nor storing messages indefinitely are practical in a system intended for large-scale deployment. One could experimentally determine an average delay δ, e.g., as the arithmetic mean of the 95 % of messages with the lowest delay, and derive a limit how long a node needs to store a message when determining matching recipients. However, this introduces other difficulties such as clients disconnecting while waiting for the context information to reach a node. Instead, the following definition of a "Localized Perfect Dissemination" relaxes the requirements of a "Perfect Dissemination" for a single node in a distributed system. The underlying assumption for this definition is the typically low delays in modern networks, while at the same time accepting a certain amount of false negatives (from a global perspective) caused by propagation delays for client contexts. A "Localized Perfect Dissemination" is also the semantics of the reference algorithm we are introducing in Section 3.3.3.

Let M_t be a contextual message that a node N received at time t, let `Contexts`(N,t) refer to the set of contexts known at node N, also at time t, and let `Origin`(C) refer to the node from which N received C.

Definition 3.8 (Localized Perfect Dissemination)**.** Let $\{N_1, \ldots, N_k\}$ be the set of ContextRouters with a direct link to node N (the *neighbors* of N) and M_t a message received at time t from node N_M. A *localized perfect dissemination* forwards a message M_t exactly once to each element of a subset $N_{M_t} \subseteq \{N_1, \ldots, N_k\} \setminus N_M$, for which the following condition holds:

$$N_i \in N_{M_t} \quad \Leftrightarrow \quad \exists C \in \texttt{Contexts}(N, t) : C \sqsubset M_t \wedge \texttt{Origin}(C) = N_i. \tag{3.2}$$

To illustrate this definition, recall our example from the beginning of the chapter: Assume that a node N has three neighbors $\{N_1, N_2, N_3\}$. Further assume that from N_3 it has received the context C shown in Table 3.1, thus informing N that N_3 can reach the client represented by this context. (Note that it is irrelevant whether C is directly registered at N_3 or whether N_3 is simply a node on the path to the access network where C is registered. We are discussing context propagation in more detail in Section 3.3.3.) When N receives the message M shown in Table 3.2 from N_1, it achieves a localized perfect dissemination by forwarding it to N_3: N_1 already has the message, whereas N does not have any information about a matching recipient that can be reached via N_2, leaving N_3 and the context C it has received from it as matching recipients.

Definition 3.8 differs from Definition 3.7 in that it takes only local information—both in space and time—into account: only contexts of which a ContextRouter has knowledge at the time it forwards a message determine to which of its neighbors to forward it. The consequence for the routing process in a distributed Contextcast system is that a context C_i must have been forwarded all the way to the source of

a message M (with $C_i \sqsubset M$) to establish the necessary knowledge on intermediate routers before routers can forward M all the way to the access network of C_i. (N.b., a client that registered C_i may still receive M, though, even if C_i wasn't forwarded all the way to the source of M. Intermediate routers may forward the message towards C_i due to a different context C_j from another ContextNode, with $C_j \sqsubset M$, until it reaches a node that already has knowledge about C_i.)

In other words, in a localized perfect dissemination, a message is only forwarded to a neighboring router if the forwarding node knows of at least one matching context that can be reached via this neighbor at the time of forwarding. These forwarded messages provides a lower bound for the message load in the system: If any one of these messages is not forwarded, one or more clients do not receive the message, even though their context matches it.

Based on this discussion, we can provide formal definitions for falsely forwarded messages.

Definition 3.9 (False negative)**.** A *false negative* on a link is a message M that is not forwarded to a neighbor N_i, even though there is at least one matching recipient C that can be reached via N_i, i.e., $\exists C \in \texttt{Contexts}(N) : C \sqsubset M \wedge \texttt{Origin}(C) = N_i$.

Definition 3.10 (False positive)**.** A *false positive* on a link is a message that is forwarded to a neighbor N_i, despite there being no matching recipient C that can be reached via N_i, i.e., $\forall C \in \texttt{Contexts}(N) \wedge \texttt{Origin}(C) = N_i : C \not\sqsubset M$. Therefore, N_i drops the message without forwarding or delivering it.

Since the Contextcast network forms an acyclic graph, for any node N and any given Context C there is only one path via which to reach C from N. The union of all these paths from N to all C_k with $C_k \sqsubset M$ is the minimum dissemination tree for a given message M. (An example for these paths and the resulting dissemination tree is shown in Figure 3.1b.) The following section introduces a reference dissemination algorithm for Contextcast, which provides a localized perfect dissemination. It also shows how a minimum dissemination tree is constructed from the local information available to the ContextRouters.

3.3.3 Directed Dissemination Reference Algorithm

In this section we present a reference algorithm to disseminate a message to the ContextNodes with matching client contexts. It employs client context information to directedly forward messages from senders to ContextNodes with matching recipients. The directed forwarding prevents the broadcasting of messages, which would limit the scalability of the system. As we are going to show, the design of the algorithm provides a localized perfect dissemination of Contextcast messages.

Context Propagation

For a localized perfect dissemination, a ContextRouter needs to know which client contexts can be reached via its neighbors. That way, the router can forward a message only to those neighbors via which a matching recipient can be reached. To this end, all context updates (newly registered, changed, and deregistered contexts) are propagated into the network. Each ContextRouter maintains a *routing table* with entries (C, N_C), recording a context C and the neighbor N_C from which it has received that context. Every incoming context update is entered in this routing table before propagating the information to all neighbors, except the one that it was originally received from. Algorithm 3.1 shows these steps in detail.

Algorithm 3.1 Client Context Propagation

Require: A context update C, received from neighbor N_C or a local client.
Ensure: (C, N_C) entered in routing table and C propagated.

```
if not a context from a local client then        ▷ Received from neighbor N_C
    Record (C, N_C) in routing table
end if
for all N ∈ Neighbors \ N_C do                   ▷ Propagate to neighbors
    Propagate a copy of C to N
end for
```

Propagating all context updates in this manner establishes a forwarding tree towards every client context. Each ContextNode is the source for the propagation of all its connected client contexts. At the same time, in the resulting forwarding tree, each client context C_i (and therefore the ContextNode with which it is registered) is a sink for all messages $M : C_i \sqsubset M$. Figure 3.1a shows an example of two such trees for contexts C_1 and C_2. The arrows next to the nodes and links denote routing table entries for the nodes, i.e., which context can be reached via which link. As one can see, each forwarding tree establishes a path from every overlay node to the one where the respective context is registered. This tree is then used when forwarding a message to establish a dissemination tree for each individual message, as we are detailing in the following section.

Message Forwarding

When a ContextRouter receives a message M_t at time t, it determines to which of its neighboring routers it needs to forward M_t, using the information in its routing table. To prevent forwarding a message twice to the same neighbor, the router maintains a list of neighbors to which it has already forwarded M_t during the

process. It iterates over every entry (C, N_C) in its routing table. If the ContextRouter has already forwarded M_t to N_C or it has received M_t from N_C, it continues with the next entry. It compares the context C from every entry with the addressing of M_t If $C \sqsubset M_t$, it forwards a copy to the neighbor N_C, from which had received C. This sequence is shown in Algorithm 3.2.

Algorithm 3.2 Message Forwarding

Require: A message M_t, received from neighbor N_M or a local client, at time t.
Ensure: M_t forwarded according to a localized perfect dissemination.

```
boolean array[neighbors] forwarded ← [false,...,false]
for all (C, N_C) ∈ Routingtable do                    ▷ Check all known contexts
    if forwarded[N_C] = true ∨ N_C = N_M then
        Continue
    end if
    if C ⊏ M_t then
        Forward a copy of M_t to N_C
        forwarded[N_C] ← true
    end if
end for
```

The forwarding Algorithm 3.2 achieves a localized perfect dissemination (cf. Definition 3.8): A message M_t is forwarded to a neighbor if and only if a ContextRouter has a routing table entry with a context C from that neighbor, such that $C \sqsubset M_t$, and if it has not received M_t from it.

While it achieves a localized perfect dissemination, matching every locally known context against a newly received message is obviously not an efficient way to determine to which neighbors to forward a message. However, efficiently determining the objects that match a set of constraints has been studied extensively in the area of databases (e.g., the Generalized Search Tree (GiST) [HNP95].) Another approach from Pub/sub systems employs an index of the constraints in subscriptions and a match-counting algorithm for forwarding notifications [CW03]. A Contextcast system can adopt a similar scheme for contexts, allowing routers to efficiently lookup the set of matching contexts and with it determine the neighbors from which it has received these contexts. Thus, for the remainder of this work, we assume that a suitably fast algorithm is used for the local forwarding decision of the routers; instead, we focus on the distribution and maintenance of the information that is needed for this decision. To not make matters more complicated than necessary, though, our algorithms are presented with the simpler approach of comparing all known contexts with a message.

Figure 3.1b shows an example of the reference algorithm forwarding a message M along the two trees for $C_1, C_2 \sqsubset M$, which were established with Algorithm 3.1 and shown in Figure 3.1a. The ContextRouters forward the message along partial dissemination trees towards the origin of each context $C_i \sqsubset M$. These partial dissemination trees form a path between the origin of M and the origin of each $C_i \sqsubset M$. The union of all these partial dissemination trees form the dissemination tree for M. Please note that, even though a link may be part of several partial dissemination trees, M is forwarded only once per link.

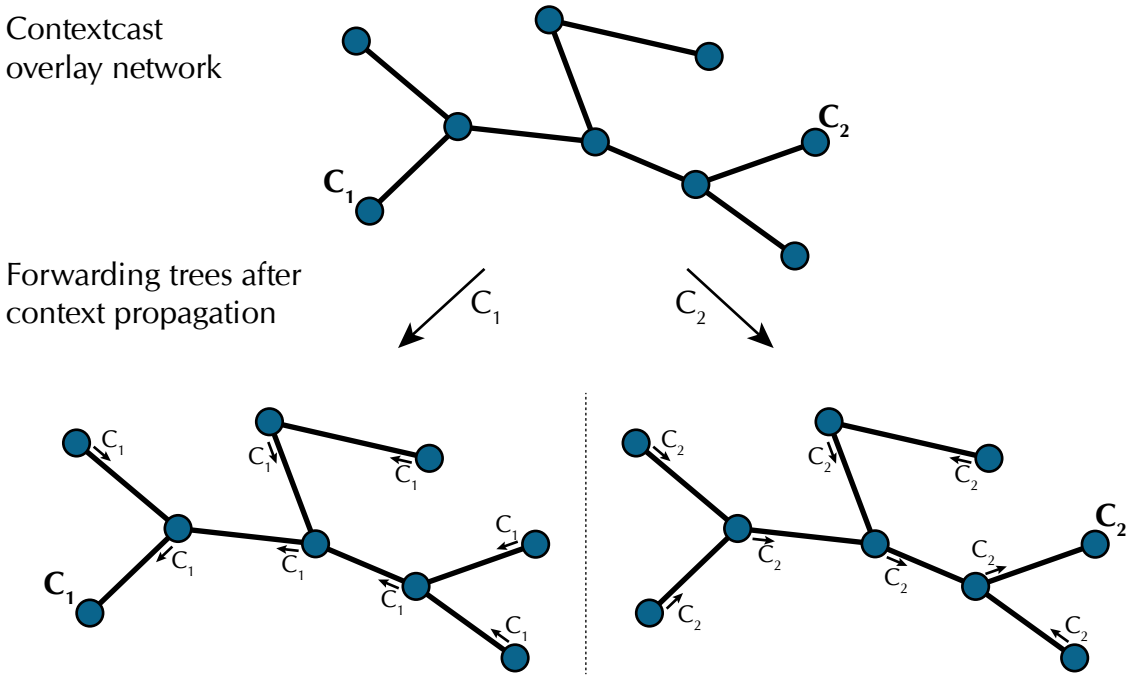
The figure also illustrates, though, that a message sent at time t might not be delivered to every matching context that was registered with the system at time t, which we mentioned in the previous section. Even with the relaxed definition of a localized perfect dissemination, this happens when a matching context C has not been propagated all the way to the sender of a message M. E.g., if the context C_2 in the figure is not propagated all the way to the origin of M when it is sent, an intermediate ContextRouter might not forward M towards C_2. This results in C_2 missing M, even though C_2 may have been registered well before the message sending time t. In the example, it is sufficient, though, if C_2 has been propagated to a node on the path towards C_1. M is forwarded to that node due to the context C_1, from where then two copies are forwarded toward C_1 and C_2, respectively.

Due to these complications in a distributed system, the routing algorithms we present implement a best-effort service for Contextcast delivery. Chapter 6 shows an approach that explicitly takes temporal aspects into account. It overcomes the limitations resulting from the propagation delay but suffers from drawbacks such as an increased overhead due to the necessary protection of privacy or an increased message delay.

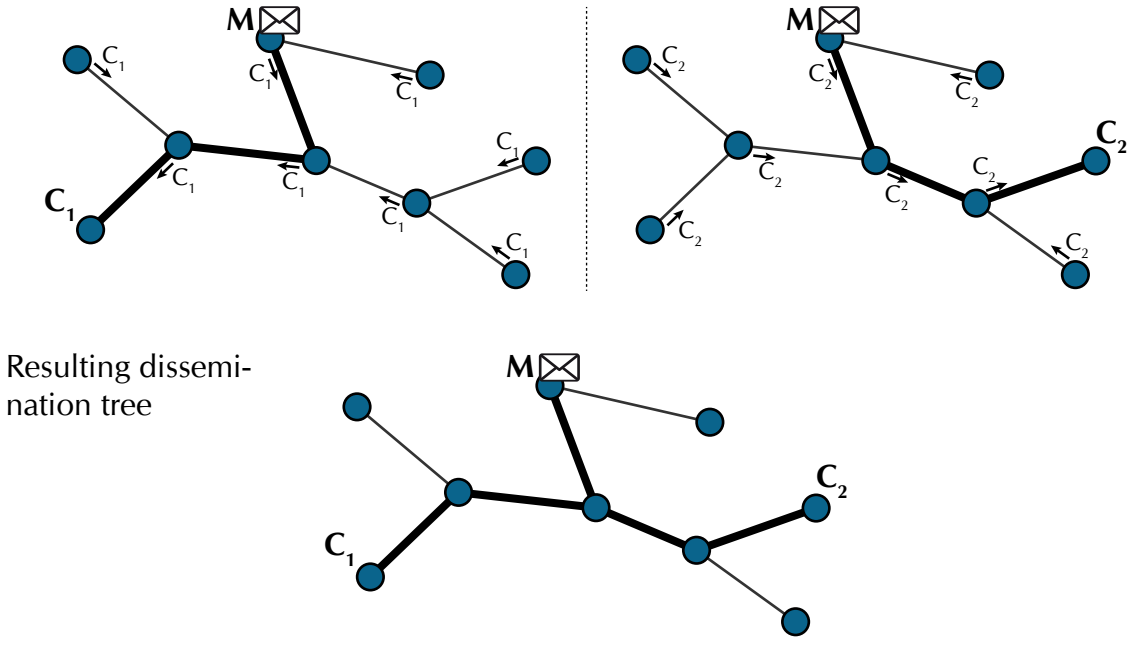
System Load

As we mentioned before, the reference dissemination algorithm indeed provides a localized perfect dissemination. I.e., a ContextRouter forwards a message only to a neighbor if there is at least one matching recipient in that direction. And, since a message is forwarded over a link at most once, it also disseminates a given message to all recipients with the minimum amount of forwarded messages in the overlay.

The process has one shortcoming, though, which limits its scalability: every context update needs to be broadcast in the Contextcast overlay network. Let us assume that clients registered with the system have an average update rate $u\frac{1}{s}$(both context registrations and deregistrations), and that there are m clients connected at a given time. Then each router must process $mu\frac{1}{s}$ updates every second. Furthermore, in a network of n routers—connected in an acyclic topology—a single context needs to be propagated to every router, causing a system-wide update load $nmu\frac{1}{s}$.



(a) Recipient-specific forwarding trees resulting from the propagation of two contexts, C_1 and C_2, registered at different ContextNodes



(b) Message forwarding of M along partial forwarding trees from two contexts C_1 and C_2, with $C_1, C_2 \sqsubset M$, and the resulting dissemination tree

Figure 3.1: Context propagation and message forwarding in Contextcast

However, while Algorithm 3.1 and Algorithm 3.2 provide a localized perfect dissemination (in the absence of failures), this update load is obviously limiting the scalability of the Contextcast system.

In Chapter 4 and Chapter 5, we therefore introduce two approaches to lower this update load and increase the scalability of Contextcast. Both are adaptive with regard to the properties of client contexts and messages that occur in the system, thus reducing the update load in the average case. The first of the two approaches, which we detail in Chapter 4, focuses on the update load caused by the number m of clients in the system. It uses coarser context information for routing, which allows routers to no longer propagate the context updates of all m clients in the network and/or reducing the contexts' update rate $u\frac{1}{s}$ because of the coarser information. The second approach in Chapter 5 is aimed at the broadcast of client contexts. Instead of propagating a context to all n routers, thus incurring an update load $nmu\frac{1}{s}$, it adaptively propagates context information only to certain routers.

The effectiveness of these approaches depends on the specific contexts and messages that occur. However, if the conditions are not favorable, both approaches gracefully degenerate to the reference algorithm introduced in this chapter. In addition, we provide experimental evaluations, which serve to determine the conditions under which the approaches work well and to provide an insight into the reduction of update load possible in a real world scenario.

Chapter 4

Directed Contextcast Forwarding using Coarse Client Context Information

> All animals are equal, but some animals are more equal than others.
>
> *(George Orwell, 'Animal Farm')*

4.1 Overview

As we have discussed in the previous chapter, maintaining complete context knowledge on every ContextRouter limits the scalability of the Contextcast system. As we are going to show in this chapter, one way to reduce this update load is to propagate coarser context information. This serves two purposes: first, it combines several client contexts into a single, coarse context to propagate, thus reducing the number m of clients that are known in the network. And second, an individual context update is propagated in the network only if the context changes sufficiently, such that it is no longer covered by the coarser representation, thus lowering the average update rate u of the clients.

Such a change needs to be transparent for applications, though, to ensure that existing applications maintain their functionality, independent of the routing optimization employed in the network. To this end, the change must not alter the semantics of the system, which we described in Chapter 3, at least from the applications' point of view.

In a system with mobile clients, this idea suggests itself particularly for the *location* attribute. This attribute may cause a steady stream of updates, depending on the sensing frequency and the accuracy of the location system, for every client context. Allowing uncertain values—i.e., areas in the case of location or, more generally speaking, ranges—reduces the load caused by such updates.

Another reason why the *location* attribute lends itself for such an approach is the system architecture with ContextNodes and their service area coverage; they

provide an inherent discretization for the *location* attribute. Thus, we first introduce the idea using the *location* attribute as an example and discuss the required changes to Contextcast for such a coarse attribute. Other attributes, however, can cause a large number of updates and thus place a high update load on the system as well. Propagating coarser information for other attributes is not as straight-forward as for *location*, though. Thus, later in this chapter, we also present a more generalized approach, which uses the similarity of client contexts to automatically determine appropriate discretizations of attributes. They are then used to aggregate similar contexts, which directly reduces the number m of context to propagate into the network.

To this end, in this chapter we first describe the requirements for a routing with coarse context information in Section 4.2. Next, we introduce two approaches to reduce the update load in the system in Section 4.3 and Section 4.4. The first one reduces the update rate for an individual attribute—*location*—and serves as a simpler example of what is required when modifying Contextcast to handle approximate information. The second one aggregates similar contexts into one update to reduce the overall number of updates. It is a generalization of the first approach that applies to arbitrary attribute types. Then, we show the results of an evaluation of these techniques in Section 4.5. After that, we discuss related work in Section 4.6, before concluding this chapter with a short summary in Section 4.7.

4.2 Requirements

From the informal description in the previous section, we can deduce a number of important requirements for using coarser client contexts in Contextcast.

Reduction of Updates. The scalability of the Contextcast system depends on the number of messages and updates that are transmitted in the network. The reference dissemination we introduced in Section 3.3.3 causes the minimum number of forwarded messages: a message is forwarded to a neighbor if and only if there is a matching recipient reachable via this neighbor. It requires a high amount of context information on every router in the network, though, and thus a large volume of updates to keep this information up-to-date. The primary goal of this approach is therefore a reduction of updates compared to the reference algorithm.

The use of coarser information for forwarding messages may introduce false positives, though. Our approach needs to balance the reduced update load from the coarse context information against the false positives it causes. Therefore, the goal is a good trade-off between the reduction of updates and increased false positives; ultimately, we aim for a reduction of overall system load, i.e., the combination of updates and false positives.

Implementation Transparency. The details of the Contextcast routing algorithms need to be transparent from an application developer's point of view. Client applications require a stable interface, regardless of the routing algorithms that are used to disseminate messages. Otherwise, any future optimization of Contextcast routing would break all existing applications.

For this reason, applications that interface with Contextcast must be able to function without modification with a routing strategy based on approximated values. In particular, applications must still be able to specify exact values when registering a context. Any optimization to use approximated values therefore must take place internally at the ContextRouters.

Preserving Delivery Semantics. Similarly to the Contextcast interface in the previous requirement, application developers must also be able to rely on the semantics of Contextcast, regardless of the routing algorithms. Unfortunately, reducing updates by propagating approximate information also means that ContextRouters make their forwarding decision on the basis of approximate information. In particular, a router may no longer be able to determine exactly to which neighbors it needs to forward a given message for a localized perfect dissemination. In this case, a message might not reach all ContextNodes with clients whose context matches it. This results in clients missing messages that actually addressed their context.

To this end, the reference dissemination from Algorithm 3.2 needs to be adapted to handle this loss of information: The general idea is to forward a message to a neighboring router if there *might* be a matching recipient that can be reached via this neighbor, based on the approximate information. Messages can always be filtered out by a downstream router if there was indeed no matching recipient.

Continuous Operation. Clients can connect to and disconnect from the Contextcast system as well as update their context at any time. This happens for different reasons, such as entering an area of bad wireless connectivity, devices switching into a standby state to preserve energy, or simply because one or more attributes change. The system needs to handle these connections, disconnections, and updates of clients efficiently. In particular, an aggregation approach must not recalculate the aggregation scheme for all contexts every time an update occurs. Otherwise, the complexity of the aggregation scheme itself would quickly overload Contextcast nodes, despite the lower number of updates propagated in the Contextcast network.

4.3 Contextcast Forwarding with Coarse Location Information

Current consumer-grade GPS hardware offers sensing frequencies of up to 5 Hz and an accuracy of 2.5 m (see [u-b11]). Propagating such exact location information into the network of ContextRouters would generate a significant amount of update load in Contextcast, even at the relatively slow pace of pedestrians of around 1.5 $\frac{\text{m}}{\text{s}}$ (see [WL89]). Since each update is effectively broadcast to all routers, these updates severely limit the scalability of the Contextcast system using the reference routing we introduced in Section 3.3.3.

One approach to reduce this update load and thus improve the scalability is to adapt a reporting protocol. Instead of the actual position of a user, such a protocol transmits a position and a guaranteed maximum deviation from it. Thus, the user's reported position never deviates more than a specified upper bound from the actual one, while potentially producing significantly less updates, depending on the actual movement of the user.

In the next section, we present a similar approach for the *location* attribute, which also guarantees an upper bound for the location error. Our approach specifically exploits the design of our system, with ContextNodes and their associated service areas. Also, the introduction of coarse *location* serves as an example for the necessary changes when we extend the support for approximated information to the other attributes in Section 4.4.

4.3.1 Service Area Approximation

The reference dissemination algorithm for Contextcast, which we presented in Section 3.3.3, relies on exact context information, including a client's *location*. This, in conjunction with contexts propagated to every ContextRouter, achieves a localized perfect dissemination. Formally, let C_t and C_{t+1} be two subsequent context updates for a client C, which contain an exact location:

$$\begin{aligned} C_t: &\quad \text{WGS84: } \mathit{location} = P_t \\ &\quad \dots \\ C_{t+1}: &\quad \text{WGS84: } \mathit{location} = P_{t+1} \\ &\quad \dots \end{aligned}$$

Both locations P_t and P_{t+1} are (exact) points in a coordinate system such as WGS84, $\mathbb{R}^2$, or $\mathbb{R}^3$. Obviously, a position that is registered with the system inherently deviates

from the actual position; it cannot always be identical to the actual position. While it is possible to reduce this deviation by using more accurate sensors and increasing the reporting frequency, this also increases the amount of updates that are propagated to the ContextRouters.

The problem of maintaining such accurate point location information in a system with mobile clients is well known from the area of Moving Objects Databases (MODs). A MOD tracks and manages the positions of a set of moving objects. To this end, moving objects transmit updates containing their location to the MOD. However, sending such updates incurs a certain communication cost in such systems. Over the years, various methods of reducing the number of such updates and therefore the corresponding communication cost have been studied (e.g., [WSCY99, LR01, ČJP05]).

One way to reduce the amount of position updates is the use of a reporting protocol such as Distance-based Reporting (DBR) [LR00, LR01]. DBR transmits a user's location $\vec{p}(t)$ at time t to the MOD. After that, it monitors the actual position $\vec{a}(t')$ of the client at time $t' > t$. If the actual position deviates more than a threshold δ from the last reported position, i.e.,

$$\left|\vec{a}(t') - \vec{p}(t)\right| > \delta,$$

the protocol sends $\vec{a}(t')$ to the server as the object's new position. Thus, with no update messages lost, DBR guarantees that a position as reported by the MOD never deviates more than δ from the actual position.

In Contextcast, clients register with a ContextNode N whose service area they are located in. For the "Localized Perfect Dissemination" from Section 3.3.3, this information is then propagated into the network, starting from N. Once a client leaves the service area of one ContextNode and enters the service area of another one, the client needs to register with the new ContextNode N_{new}. This information is then again propagated into the network, informing ContextRouters that this particular client can now be reached via N_{new}. Thus, moving from one service area to another one requires that an update is broadcast into the network. (In fact, it requires two updates, one deregistering the context starting from the old ContextNode, N, and one registering at the new one, N_{new}.) These context updates when leaving one service area and entering another one are mandatory, since they inform the routers about a new attachment of a client at a ContextNode.

We adopt the DBR approach for Contextcast, but adapt it to our particular system design: we use the service area as the reporting threshold and thus the tolerated uncertainty. Thus, instead of propagating exact *location* information, the system propagates a modified context C' with a coarse *location*, which corresponds to the ContextNode's service area A_1 (with A_1 specified as a polygon of WGS84

coordinates, e.g.).

$$C'_t: \quad \text{WGS84: } location = A_1$$
$$\ldots$$

As long as a client remains within one service area, there is no need to update the client's context, at least not due to movement. At some time t', when the client moves out of the old and into a new service area A_2 and thus connects to a different ContextNode, an update is required anyway.

$$C'_{t'}: \quad \text{WGS84: } location = A_2$$
$$\ldots$$

The ContextNodes are responsible to replace an exact *location* with the approximated one to ensure that the optimization is applied to unmodified clients as well. Newer clients can directly register an approximate *location* if that is desired. In that case, a ContextNode does not replace the already approximate information. The complete process, shown in Algorithm 4.1, fulfills the first two requirements as well as the fourth one we listed in Section 4.2. First, it reduces the rate of updates in the system, since movement-related updates are no longer propagated as long as a client remains within one service area. The only movement-related updates that are still sent are those that occur when a client moves from one ContextNode to another. Second, it does not require any changes to the clients. The replacement of an exact *location* attribute with the approximation of the current ContextNode's service area takes place transparently on said ContextNode. Clients can continue to register and update contexts containing exact *location* information with a ContextNode. However, while these context updates with exact *location* remain in a service area and are not propagated into the network, they still consume power on the client and may congest a shared wireless medium. Therefore, it is desirable to enable (future) clients to also directly register an approximated *location* of the current service area to reduce power and bandwidth consumption in the access networks. Third, the replacement of a context's location happens once for every new context update that is sent to a ContextNode, without affecting any other client context. Also, it does not affect context invalidations that are propagated into the network. Thus, the approach also fulfills the requirement of an efficient continuous operation.

In the next section, we show how the forwarding algorithm has to be adjusted to fulfill the third requirement, a preservation of the delivery semantics despite using approximate *location* information for the forwarding decision.

Algorithm 4.1 Client Context Propagation with Coarse Location

Require: A context update C_u, received from neighbor N_C or a local client; the service area A_{local} in case of a ContextNode.
Ensure: (C_u, N_C) entered in routing table and C propagated.

```
if not a context from a local client then          ▷ Received from neighbor N_C
    Record (C_u, N_C) in routing table
else                                               ▷ Received locally
    if C_u contains location attribute ∧ location is exact then
        Replace location with: location = A_local
    end if
end if
for all N ∈ Neighbors \ N_C do                     ▷ Propagate to neighbors
    Propagate a copy of C_u to N
end for
```

4.3.2 Forwarding

As we have argued in Section 4.2, the delivery semantics of Contextcast needs to be preserved when optimizing the routing strategy. The reference forwarding we presented in Algorithm 3.2 evaluates a message's constraints against all known contexts to determine the neighboring routers that a message needs to be forwarded to. It delivers a message to all neighbors that can reach one or more matching recipients. Since ContextRouters no longer have exact values for a client's *location* attribute, the forwarding algorithm must ensure that clients with matching contexts do not miss messages due to the optimization. To this end, the evaluation of location constraints needs to be modified to handle the approximate location and still match all the client contexts it would have matched if the routers had exact location information. The necessary changes are rather straight-forward, though, as the following example shows.

First, let C_1 and C_2 represent two clients, who have each registered a context with an exact *location* attribute of position P_1 and P_2, respectively. Both P_1 and P_2 are simple WGS84 latitude, longitude, and altitude coordinates. P_1 and P_2 are within the service area of the same ContextNode, to which both C_1 and C_2 are connected. (For

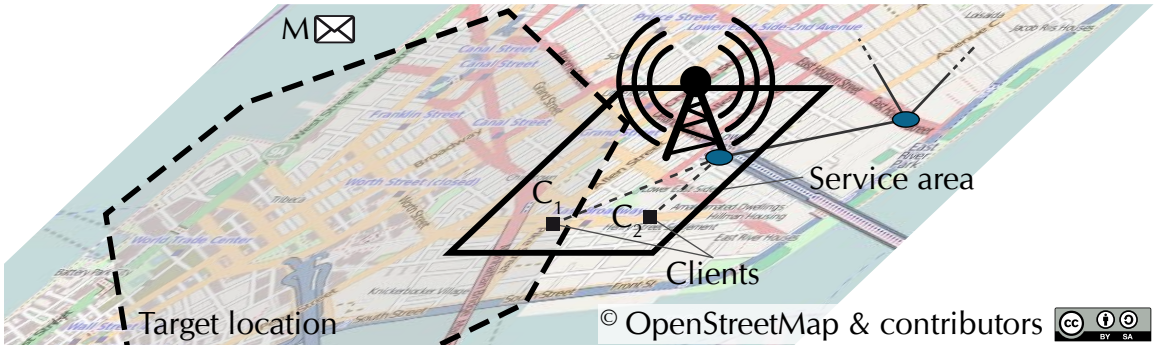


Figure 4.1: Evaluating *location* constraints with approximate client *location*s

clarity, we have omitted other context attributes of C_1 and C_2 from this example.)

$$C_1: \quad \text{WGS84: } \mathit{location} = P_1$$
$$\ldots$$

$$C_2: \quad \text{WGS84: } \mathit{location} = P_2$$
$$\ldots$$

Second, let M be a message that contains a constraint on the target location $\phi_{location}$. (Again, we have omitted other constraints from M, since they provide no additional insights in the example. We assume that both C_1 and C_2 match any constraints in M other than the one on *location*.)

$$M: \quad \text{WGS84: } \mathit{location} \in \text{<polygon of target location>}$$
$$\ldots$$

Thus, whether the two contexts match the constraints of M is determined by the *location* constraint alone. Let C_1 be within the target location of M, while C_2 is outside. This means that $C_1 \sqsubset M$ and $C_2 \not\sqsubset M$. This situation is also depicted in Figure 4.1.

The information of C_1 and C_2 is propagated to the ContextRouters. As we described in Section 4.3.1, the ContextNode replaces the exact *location* in C_1 and C_2 with its own service area A_1 before propagation—creating C_1' and C_2':

$$C_1': \quad \text{WGS84: } \mathit{location} = A_1$$
$$\ldots$$

$$C_2': \quad \text{WGS84: } \mathit{location} = A_1$$
$$\ldots$$

Thus, the ContextRouters have to make their forwarding decision based on the approximate information in C_1' and C_2'. The *location* constraint in M has not changed, however.

To achieve that every client with $C \sqsubset M$ receives M, nodes need to evaluate the *location* constraint in messages differently. According to Definition 3.4, the *location* constraint of M, $\phi_{location}$: *location* $\in$ <polygon of target location>, is evaluated for the contexts C_1 and C_2 as

$$P_1 \in \text{<polygon of target location>} \rightarrow \texttt{true}$$

and

$$P_2 \in \text{<polygon of target location>} \rightarrow \texttt{false}.$$

For a point location such as P_1 and P_2, this is simply a test whether the point is contained within the described area, i.e., the polygon.

However, with client positions approximated by an area, the evaluation becomes more complicated. Thus, evaluating the constraint for C_1' and C_2', with the approximate *location* A_1, in both cases results in the expression

$$\begin{aligned} &\textit{location} \in \text{<polygon of target location>} \\ &\Leftrightarrow A_1 \in \text{<polygon of target location>}. \end{aligned}$$

Even if the definition of the operator $\in$ for on location is extended to apply to two areas instead of a point and an area, this example shows that the distinction between C_1 and C_2 is lost: Since both C_1' and C_2' result in the evaluation of the same expression, either both yield `true` or both yield `false`.

Obviously, since $C_1 \sqsubset M$, the result of the evaluation must be `true`, otherwise C_1 would miss this message. Unfortunately, if C_1 were not present and a ContextRouter evaluates the same expression for C_2', they forward M anyway, despite $C_2 \not\sqsubset M$. This is an example of an unnecessarily forwarded messages due to the information lost in the service area approximation.

To achieve that this expression indeed yields `true`, we extend the definition of the operator $\in$ of the *location* attribute as follows:

Definition 4.1 (Evaluation of operator $\in$ with coarse *location*)**.** Let A_1, A_{target} be two areas, e.g., A_1 is a ContextNode's service area (and thus also used as coarse *location* in client contexts), A_{target} is the target *location* of a message. The comparison $A_1 \in A_{\text{target}}$ is then evaluated as

$$\begin{aligned} A_1 \in A_{\text{target}} &\Leftrightarrow \exists P : P \in A_1 \wedge P \in A_{\text{target}} \\ &\Leftrightarrow A_1 \cap A_{\text{target}} \neq \emptyset. \end{aligned}$$

In other words, with coarse location attributes, the evaluation results in `true` if and only if there is a possible location within the coarse client *location* that would match the constraint, i.e., there is a non-empty intersection of the two areas.

In the following section, we show how this concept of coarser values for individual attributes can be extended to attributes other than *location*.

4.4 General Aggregation of Client Context Information

Reducing the update rate of client contexts using coarse information is similar to approaches that have been researched in the area of Pub/sub systems: routing schemes with filter *covering* and *merging* [Müh02]. The idea behind these two concepts is to propagate only a very general subscription in the system and thus reduce both the number of subscriptions and their update rate. (The key difference between the two is the origin of the the propagated, coarse subscription: informally speaking, given a set of client subscriptions, *covering* selects one of them that is the superset of the other ones, whereas *merging* creates a new, previously non-existent subscription, as the superset of the given client subscriptions. For a more detailed discussion of *covering* and *merging*, refer to Section 4.6.)

Approximating a client's *location* with the service area of the ContextNode it is connected to resembles a routing scheme with *covering* in Pub/sub systems. The approach is limited to a single attribute with particular properties, though. First, movement usually causes small, localized changes in *location*: a client rarely disappears in one place and reappears far away from its last position. As long as a client maintains a connection to the system, their *location* changes little between updates (at least if the *location* is described using a geometric location model such as WGS84). (The obvious exception to this is a client who is disconnected from the Contextcast network for the duration of a journey, e.g., when traveling by plane; such a situation occurs relatively rarely, though.) Second, the responsibility of ContextNodes for a defined service area inherently provides a discretization of the attribute value. All clients that are within the same service area are in close proximity to each other. This discretization serves as a readily-available approximation of clients' positions.

Other attributes in Contextcast do not necessarily offer the same properties. An attribute *transport*, which describes a client's method of transportation, can change from "/pedestrian" to "/public/metro" to "/motorized/car", all during their way home from work. Due to these potentially large changes, it is rather difficult to approximate such a value. Also, it is unclear how such an approximated value would look like. But even for an attribute such as *age*, for which an approximation might seem easier, the answer is not that obvious. Assume, e.g., four client contexts,

with an *age* attribute of 22, 24, 28, and 30 years, respectively. Should the system propagate them (informally) as "some clients between 20 and 25" and "some clients between 25 and 30"? Or rather as "some clients between 20 and 30"?

To solve these challenges, we are going to show an aggregation approach that determines similar contexts and propagates an aggregated view of these contexts into the network. This is a generalization of the *merging* routing scheme used in Pub/sub systems, applied to Contextcast, as we are discussing in Section 4.6. The assumption behind these *aggregated contexts* is that clients are not all completely different. Whenever there are similarities between some clients, this can be exploited to combine the information of two or more clients. We also assume that such similarities accumulate at certain locations. On a university campus, most of the clients will be of type student; they are also all of a similar age. For each academic major, there is a set of clients who have chosen this major, etc. Similar arguments can be made for other locations.

An aggregation of client contexts is beneficial to the scalability of Contextcast in two ways: First, an aggregation of contexts reduces the number of contexts that are propagated to the routers. It combines similar contexts into an aggregated view and propagates this coarser representation to the routers. Second, like the *location* approximation using a ContextNode's service area, such an aggregation also reduces the rate of context updates that are propagated into the network. If a context is propagated as an aggregated context, any update of this context only needs to be propagated if it falls outside of the aggregation it is part of. For the *location* attribute, this means that a context moves outside of the approximated area, i.e., the ContextNode's service ares. In the example of the *age* attribute from before, if the oldest client turns 31, its context update falls outside of the aggregation (regardless of which of the two example aggregations was in fact chosen). For other attributes, this depends on the actual aggregation, as we are showing in the following sections.

While this approach reduces the number of contexts that need to be propagated to the ContextRouters as well as having lower update rates for the aggregated contexts, such an aggregation introduces false positives into the system. The reason is the same as for coarse location, ContextRouters may no longer be able to evaluate exactly whether a given aggregation contains a client context that matches a message. Thus, the effectiveness of this approach depends strongly on the similarity of contexts and the precision of the resulting aggregations. They determine how precise the forwarding of messages is with regard to the actual registered client contexts. Section 4.5.2 contains an experimental evaluation of the effects of various conditions on the update and false positive load in the Contextcast system.

Conceptually, aggregating client contexts in Contextcast is similar to filter covering and merging in Pub/sub systems. We discuss this relationship in more detail in Section 4.6.

4.4.1 Aggregated Context Information

The main idea behind the aggregation of client contexts is a reduction of update load by (1) reducing the update rate through coarse values, and (2) combining several client contexts into one aggregated context. For the discussion of our aggregation approach, we distinguish two types of contexts: First, the contexts that clients register with a ContextNode are called *singleton contexts*. Such a singleton context precisely describes the context of an individual client, before any approximation or aggregation. Second, *aggregated contexts* result from an aggregation of a number of contexts. These can either be singleton contexts, other, previously aggregated contexts, or a mixture of both.

Both singleton and aggregated contexts describe clients in our system and can therefore be used with the reference forwarding algorithm from Section 3.3.3. However, similarly to the handling of approximated *location* information, the matching of messages with aggregated contexts must be adapted due to the uncertainty that an aggregation introduces. The necessary changes depend on the actual method of aggregating contexts. We are showing such an aggregation method as well as the changes to the matching algorithm in section 4.4.2. With this approach, instead of propagating a large number of singleton contexts, the system can propagate a smaller number of aggregated contexts to represent a set of singleton contexts.

An abstract way to define such an aggregation is in terms of the addressed clients: If at least one of a set of contexts matches a given message then the aggregation of these contexts must match the message as well. This ensures that a client does not miss a message whose addressing matches its context due to an optimized routing algorithm based on aggregated contexts (a "false negative", see Definition 3.9). This fulfills our requirement that an optimization to the routing algorithm must not alter the semantics of Contextcast (see Section 4.2). Formally, an aggregation of client contexts is defined as follows:

Definition 4.2 (Aggregation of client contexts)**.** Let C be a set of Client Contexts $\{C_1, \ldots, C_n\}$. An *Aggregation* C' of these contexts is a set of contexts $\{C'_1, \ldots, C'_k\}$ (where typically $k \ll n$, to achieve a reduction in update load) for which the *Aggregation Condition* holds.

Definition 4.2 is on purpose formulated in a way that abstracts from the actual method used to compute an aggregation. All that is required is that the Aggregation Condition in Definition 4.3 holds for such a set of contexts.

When aggregating contexts, singleton or composite, information is usually lost. A larger number of client contexts is represented by a smaller number of aggregated client contexts. This also affects the forwarding decisions of ContextRouters. The following Aggregation Condition, however, ensures that such an information loss

does not lead to clients missing messages even though their context matches a message (i.e., *false negatives*).

Definition 4.3 (Aggregation Condition)**.** Let $C = \{C_1, \ldots, C_n\}$ and $C' = \{C'_1, \ldots, C'_k\}$ be two sets of contexts and let M be an arbitrary Contextcast message. We call C' an aggregation of C if and only if the following condition holds:

$$\forall M(\exists C_i \in C : C_i \sqsubset M) \rightarrow (\exists C'_j \in C' : C'_j \sqsubset M).$$

In other words, if a context in C matches M, at least one context in the aggregation C' must also match M. This ensures that a router forwarding on the basis of aggregated information C' forwards to at least the same neighbors as with the unaggregated information C.

As a downside of the aggregation of client contexts, however, ContextRouters can no longer achieve a localized perfect dissemination, due to the loss of information. Before, with exact context information, a message was forwarded to a neighbor if and only if there was at least one matching recipient reachable via this neighbor. With aggregated client contexts, the ContextRouters perform forwarding based on the aggregated information that was propagated to them. Because of the Aggregation Condition, all that is required for an aggregation is that an unaggregated context $C_i \in C : C_i \sqsubset M$ is also represented in an aggregation such that $\exists C'_j \in C' : C'_j \sqsubset M$.

The aggregation of contexts is therefore a trade-off between a reduction of update load and an increase in message load due to these false positives. To lessen the impact of these additional false positives, we base our aggregation strategy on the similarity of client contexts. This ensures that very little information is lost when aggregating contexts. Section 4.5.2 shows that the reduction in updates outweighs the additional load from false positives in our scenario.

After providing the basic definitions of our context aggregation approach, the next section shows an algorithm which provides an aggregation for a pair of contexts. As we argued before for approximated location data, the matching of such aggregated contexts against messages needs adaptation. Thus, we are also showing the necessary changes for the evaluation of constraints with aggregated contexts.

4.4.2 Pairwise Context Aggregation

Based on the discussion in the previous section and the matching semantics we introduced in Section 3.2, we can now present an approach to aggregate client contexts. It is designed to aggregate two contexts into one, a decision that was influenced by the Continuous Context Aggregation algorithm, which we present in Section 4.4.4.

Aggregation

The Pairwise Context Aggregation (PCA) shown in Algorithm 4.2 aggregates two arbitrary contexts into one. The algorithm achieves this in two steps: First, every

Algorithm 4.2 Pairwise Context Aggregation

Require: Two contexts C_1 and C_2 (singleton contexts or already aggregated ones).
Ensure: An aggregated context C_{aggreg}.

```
function AGGREGATEPAIRWISECONTEXTS(C_1, C_2)
    C_aggreg ← ∅
    for all α_{i,1} ∈ C_1 do                ▷ Add attributes that are only in C_1 or in both
        if ∃α_{j,2} ∈ C_2 : type_{α_{i,1}} = type_{α_{j,2}} ∧ name_{α_{i,1}} = name_{α_{j,2}} then
            α_merged ← (type_{α_{i,1}}, name_{α_{i,1}}, value_{α_{i,1}} ∪ value_{α_{j,2}})
            C_aggreg ← C_aggreg ∪ α_merged
        else                                ▷ Attribute only present in C_1
            C_aggreg ← C_aggreg ∪ α_{i,1}   ▷ Insert in aggregated context unmodified
        end if
    end for
    for all α_{j,2} ∈ C_2 do                ▷ Add remaining attributes that are only in C_2
        if ∄α_k ∈ C_aggreg : type_{α_{j,2}} = type_{α_k} ∧ name_{α_{j,2}} = name_{α_k} then
            C_aggreg ← C_aggreg ∪ α_{j,2}
        end if
    end for
end function
```

attribute that occurs in only one of the contexts C_1 and C_2 is added to the aggregated context without modification. Second, for every attribute that occurs in both C_1 and C_2, it merges the values and adds the merged attribute to the aggregated context.

Obviously, this merging of attribute values needs to be defined for the different attribute types in a Contextcast system. We are showing example merging strategies for important attribute types, which we also discussed in Section 3.2.1. These were also used in our prototype in Section 4.5.2. Together with the discussions in the previous sections, these can be easily adapted to other attribute models if desired.

- For example, for WGS84 *locations*, a merged value can be the bounding rectangle of both positions. Or, as a more precise alternative, a polygon that closely approximates the original positions or areas.
- For ordered, numerical types, the merged value can be the set of all values. Alternatively, if the number of elements is large and their density high, it can be a closed interval from the smallest to the largest value.

- Similarly, for an attribute based on a tree of values, such as a type hierarchy, a merged value can be the set of all values. Or, for a large number of dense elements, it can be a common ancestor of the nodes in the tree.

Since the algorithm can aggregate both singleton and already aggregated contexts, the definition of such a merging of values must take care that it also supports values that resulted from a previous pairwise aggregation. For example, for an ordered, numerical type, the result of merging an interval $[v_1, v_2]$ with an individual value v_3 depends on the value of v_3:

$$\texttt{merge}\left([v_1, v_2], v_3\right) = \begin{cases} [v_1, v_3] & \text{for } v_2 \leq v_3 \\ [v_3, v_2] & \text{for } v_3 \leq v_1 \\ [v_1, v_2] & \text{for } v_1 < v_3 < v_2 \end{cases}$$

Constraint Matching for Aggregated Contexts

Similarly to the approximate *location* we introduced in Section 4.3, ContextRouters must be able to handle such merged attribute values when evaluating message constraints. This depends on the actual merging that takes place when aggregating contexts. Based on the discussion in that section and the example merging strategies we provided, we are showing the changes that are necessary here.

Similarly to the evaluation of the inexact *location* of the service area approximation, constraints on such merged attribute values must yield `true` if there might exist a value in the merged attribute that fulfills the constraint.

- We have already discussed the evaluation of a *location* attribute that is specified as a region instead of a point in Definition 4.1.

- For an ordered, numeric attribute with a set of values this is the case if the constraint evaluates to `true` for any element of the set. If the value is specified as a range of values, a constraint yields `true` if there is a value within the range that evaluates to `true` for a given constraint.

- For an attribute based on a hierarchy, with a value represented as a set of nodes in the hierarchy, a constraint yields `true` if it yields true for any of the values in the set. If the merged value is a common ancestor instead, there is no simple universal answer. The evaluation of the constraint depends on the comparison operator and its semantics for that attribute. The designer of the attribute system, in this case, needs to specify the evaluation of all constraint operators when dealing with aggregated values.

An application of Algorithm 4.2 to two contexts, together with the changes to matching discussed here, results in an aggregation according to Definition 4.2. I.e., the resulting context fulfills the Aggregation Condition from Definition 4.3.

Proof. Let C_1 and C_2 be two contexts that are used as input to Algorithm 4.2, and C_{aggreg} the result of this algorithm. Furthermore, let M be a message with addressing $\{\phi_1, \ldots, \phi_k, \phi_{k+1}, \ldots, \phi_l\}$ $(1 \leq k \leq l)$ and, without loss of generality, let $C_1 \sqsubset M$. (N.b. nothing is said whether $C_2 \sqsubset M$.) The constraints $\{\phi_1, \ldots, \phi_k\}$ are constraints on attributes that occur only in C_1, $\{\phi_{k+1}, \ldots, \phi_l\}$ are constraints on attributes that occur in C_1 and also in C_2. (Obviously, according to Definition 3.4 and Definition 3.5, since $C_1 \sqsubset M$ there must be a matching attribute in C_1 for every constraint in M. Some of the attributes, however, may also appear in C_2.)

Algorithm 4.2 adds all attributes that occur only in C_1 to C_{aggreg} unmodified. Therefore, since $C_1 \sqsubset M$, $\forall \phi \in \{\phi_1, \ldots, \phi_k\} : C_{\text{aggreg}} \sqsubset \phi$. (Similarly, it also adds all attributes that occur only in C_2 to C_{aggreg} unmodified. They are, however, irrelevant for the purpose of this proof.)

Furthermore, Algorithm 4.2 adds all attributes that occur in both C_1 and C_2 to C_{aggreg} as well; their values, however, are merged to represent the values that were present in the two original contexts. The merged value for such an attribute contains, in particular, the value of the corresponding attribute in C_1. Also, the constraint evaluation was adjusted accordingly, such that $\forall \phi \in \{\phi_{k+1}, \ldots, \phi_l\} : C_{\text{aggreg}} \sqsubset \phi$ holds.

Therefore, $\forall \phi \in \{\phi_1, \ldots, \phi_l\} : C_{\text{aggreg}} \sqsubset \phi$ and thus $C_{\text{aggreg}} \sqsubset M$. □

The algorithm can aggregate a set of arbitrary contexts into a single aggregated context according to Definition 4.2. To this end, the system repeatedly aggregates two contexts until only one remains. Since the result of each of these repeated pairwise aggregation fulfills Definition 4.3, it holds that the result of its repeated application to a set of contexts also fulfills Definition 4.3.

PCA is the basis of the aggregation strategy that we present in the remainder of this section.

4.4.3 Aggregation Selection: Context Similarity

While the repeated application of the pairwise context aggregation from Algorithm 4.2 aggregates a given set of contexts into a single one, it obviously needs to be applied to similar contexts: For example, if we aggregate a context with an *age* of 8 with one with an *age* of 80, the resulting information loss might be enormous (a context with *age* $= [8, 80]$). This would lead to a large amount of *false positive* messages, namely all messages addressed to an *age* between 8 and 80. Regarding the *location* attribute, a similar argument can be made for aggregating the contexts

of one client in Europe and one client in Australia. Therefore, the question remains "how to select a good set of (similar) contexts for aggregation?" This requires a means to measure the similarity between client contexts.

Looking at the application of Algorithm 4.2, we can identify two types of uncertainty that it introduces into aggregated contexts. Both of them can cause false positives during the forwarding decision:

1. *New attribute combinations*: Attributes that previously did not occur together in one context may end up together in an aggregated context. However, all constraints in a message must match for a message to be forwarded and ultimately delivered to clients. Therefore, these additional combinations can lead to aggregations matched by messages that would not have matched the individual contexts.

   For example, let $C_1 = \{\alpha_1\}$ and $C_2 = \{\alpha_2\}$ be two contexts, and $C_{\text{aggreg}} = \{\alpha_1, \alpha_2\}$ be the result of aggregating the two. Also, let M be a message with constraints on both α_1 and α_2, i.e., $M = \{\phi_1, \phi_2\}$. Further, each of the attributes in C_1 and C_2 matches the corresponding constraint. But $C_1 \not\sqsubset M$ and $C_2 \not\sqsubset M$, since each context is missing one of the two attributes. However, $C_{\text{aggreg}} \sqsubset M$, due to the newly introduced combination of α_1 and α_2 in one context.

2. *Uncertain values*: Attributes that occur in two or more contexts have their values merged. This may create new values for attributes, which were not present in any of the individual contexts. Again, this can lead to constraints matching an aggregated attribute value, which would otherwise not have matched the individual, unaggregated values. For example, let $C_1 = \{\alpha_1 = \text{value}_1\}$ and $C_2 = \{\alpha_1 = \text{value}_2\}$ be two contexts, and $C_{\text{aggreg}} = \{\alpha_1 = [\text{value}_1, \text{value}_2]\}$ be the result of aggregating the two. Also, let M be a message with a constraint on α_1, i.e., $M = \{\phi_1 = \text{value}_3\}$, with $\text{value}_1 < \text{value}_3 < \text{value}_2$. Then $C_1 \not\sqsubset M$ and $C_2 \not\sqsubset M$, due to the values of the attribute. However, $C_{\text{aggreg}} \sqsubset M$, due to the newly introduced values in the range $[\text{value}_1, \text{value}_2]$.

We therefore propose to measure the overall similarity of client contexts by two components, a structural similarity for their respective attribute sets and a value similarity for the values of the attributes occurring in both contexts.

Structural Similarity

The structural similarity measures how similar two contexts are in terms of their attribute combinations. We adopt the well-known and widely used Jaccard index J (see [Jac01]) to express this similarity. It defines the similarity of two sets A and B as

the number of elements that occur in both over the total number of distinct elements in both sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

With this definition, we can measure the structural similarity of two contexts using the number of attributes shared by the two contexts over the total number of attributes:

Definition 4.4 (Structural Similarity)**.** Let $\texttt{Attributes}(C)$ denote the set of attributes (without values) that occur in a context C. The *structural similarity* $S_{\text{structural}}$ between two contexts C_1 and C_2 can then be expressed as

$$S_{\text{structural}}(C_1, C_2) = \frac{|\texttt{Attributes}(C_1) \cap \texttt{Attributes}(C_2)|}{|\texttt{Attributes}(C_1) \cup \texttt{Attributes}(C_2)|}. \tag{4.1}$$

Obviously, for $\texttt{Attributes}(C_1) = \texttt{Attributes}(C_2)$, i.e., C_1 and C_2 containing the same set of attributes, albeit with probably different values, this results in $S_{\text{structural}}(C_1, C_2) = 1$, i.e., the highest structural similarity. In contrast, $\texttt{Attributes}(C_1) \cap \texttt{Attributes}(C_2) = \emptyset$, i.e., C_1 and C_2 do not share any attribute, results in $S_{\text{structural}}(C_1, C_2) = 0$, i.e., the lowest structural similarity.

Value Similarity

As its name implies, the value similarity measures how similar two contexts are in terms of the values of their attributes. For point values of an attribute with a defined distance measure the similarity can easily be defined, e.g., as the reciprocal of the distance between two values. This can easily be extended for objects that contain numerous attributes with point values, which are often used in data analysis. In this case, the objects are numerical vectors and their (dis)similarity can be expressed, e.g., by a distance measure in $\mathbb{R}^n$. (See, e.g., [JMF99] or [XWI05] for an overview of clustering approaches.)

This concept cannot be easily applied to the similarity of contexts in our Contextcast system, though. First, two contexts need not be defined on the same set of attributes. Second, client contexts can contain attributes that do not have a meaningful distance measure defined, such as a type hierarchy, or whose values are not simple points but value ranges. Contexts in Contextcast are therefore more similar in nature to the *symbolic objects* described by Gowda and Diday in [GD92]. The attribute types in Contextcast, in particular, are very similar to the classification they provide. We have therefore adapted their classification of different attribute types from [GD92] as well es their defined similarity measures:

- Quantitative Attributes

  - continuous ratio values, e.g., velocity.
  - discrete absolute values, e.g., age.
  - interval values, e.g., duration.
- Qualitative Attributes
  - nominal (unordered), e.g., color or gender.
  - ordinal (ordered), e.g., designation.
- Structured Attributes
  - tree-ordered sets, e.g., a type hierarchy.

Generally, the system we present can support arbitrary types. All that is required is that the designer of the attribute model provides a definition of both a similarity measure for a type's values (i.e., a function $(v_1, v_2) \mapsto [0;1]$) and an aggregation scheme for the values.

The similarity of two values of an attribute depends on the respective class of the attribute. The authors in [GD92] and subsequently [GR95] describe a method to compute the similarity for the attributes of these types, with the exception of structured variables, which they mention as one possible type but never detail. From this information, they then compute the similarity between two symbolic objects by adding the similarity values of all attribute-wise similarities.

We closely follow the approach in [GR95] for the attribute similarity of client contexts. However, there are three notable differences: First, Gowda et al. mention that the composite objects need not be defined on the same set of attributes but never detail how it affects the computation of similarity. As we have shown in Section 4.4.3, the attribute sets that make up two contexts are incorporated in the computation of the structural similarity in Contextcast. We show later in this section how to use both the structural and value similarity of two contexts to compute their overall similarity. Second, they simply add the value similarities of all attributes that occur in both objects to calculate the resulting similarity of the objects. Obviously, this leads to higher similarities the more attributes two objects share. As we are discussing later in this section, this runs contrary to an intuitive notion of similarity. Third, they do not describe a similarity measure for structured attributes, which are an important part in our system. We therefore provide a similarity definition for this type of attributes.

According to [GR95], independent of the type of an attribute α_i, we can compute the value similarity as the sum of three components:

- $S_{\text{position}}(\alpha_{i,1}, \alpha_{i,2})$: Similarity due to position

- $S_{\text{span}}(\alpha_{i,1}, \alpha_{i,2})$: Similarity due to span
- $S_{\text{content}}(\alpha_{i,1}, \alpha_{i,2})$: Similarity due to content

The similarity due to *position* is used to describe the relative position of two values on a real line. The component due to *span* measures the relative size of two attribute values, without considering common parts. The similarity due to *content* indicates the common parts between two values.

The similarity components for *position*, *span*, and *content* are all in $[0, 1]$. However, a simple addition of the three components often results in a similarity $S > 1$. In particular, since not all attribute types employ all three components to measure similarity, this would distort the results for different attribute classes. We therefore propose a small change, which defines the value similarity of a single attribute α_i as the arithmetic mean of these three components:

$$S_{\text{value}}(\alpha_{i,1}, \alpha_{i,2}) = \frac{S_{\text{position}}(\alpha_{i,1}, \alpha_{i,2}) + S_{\text{span}}(\alpha_{i,1}, \alpha_{i,2}) + S_{\text{content}}(\alpha_{i,1}, \alpha_{i,2})}{3}. \quad (4.2)$$

Thus, the value similarity of a given attribute is also in $[0, 1]$, independent of whether a particular attribute class uses only one, two, or all three of the above components.

Quantitative Attributes. According to [GD92, GR95], the similarity of quantitative *interval* types can be computed from the similarity due to *position*, *span*, and *content*:

The similarity due to *position* is defined as

$$S_{\text{position}}(\alpha_{i,1}, \alpha_{i,2}) = 1 - \frac{|\alpha_{i,1,l} - \alpha_{i,2,l}|}{U_{\alpha_i}},$$

with $\alpha_{i,1,l}, \alpha_{i,2,l}$ the lower bounds of $\alpha_{i,1}$ and $\alpha_{i,2}$, respectively, and U_{α_i} the maximum interval of the attribute α_i.

Similarity due to *span* is defined as

$$S_{\text{span}}(\alpha_{i,1}, \alpha_{i,2}) = \frac{l_{\alpha_{i,1}} + l_{\alpha_{i,2}}}{2 \cdot l_s},$$

with $l_{\alpha_{i,1}}, l_{\alpha_{i,2}}$ the length of the intervals $\alpha_{i,1}$ and $\alpha_{i,2}$, respectively, i.e., the difference of the upper and lower bound; and l_s the span length of intervals $\alpha_{i,1}, \alpha_{i,2}$, i.e., $l_s = |\max(\alpha_{i,1,u}, \alpha_{i,2,u}) - \min(\alpha_{i,1,l}, \alpha_{i,2,l})|$, where $\alpha_{i,1,u}, \alpha_{i,2,u}$ are the upper bounds of $\alpha_{i,1}, \alpha_{i,2}$, respectively.

The similarity due to *content* is defined as

$$S_{\text{content}}(\alpha_{i,1}, \alpha_{i,2}) = \frac{\text{inters}}{l_s},$$

with inters the length of the intersection of $\alpha_{i,1}$ and $\alpha_{i,2}$.

The resulting normalized similarity for quantitative interval types is therefore

$$S_{\text{value}}(\alpha_{i,1}, \alpha_{i,2}) = \frac{S_{\text{position}}(\alpha_{i,1}, \alpha_{i,2}) + S_{\text{span}}(\alpha_{i,1}, \alpha_{i,2}) + S_{\text{content}}(\alpha_{i,1}, \alpha_{i,2})}{3}. \quad (4.3)$$

Figure 4.2 illustrates these three components with two quantitative interval type attributes, age_1 and age_2. Let the attribute *age* be an integer attribute with a value range $0 \leq x \leq 120$. Also, let $age_1 = \{x \in \mathbb{Z} \mid 23 \leq x \leq 26\}$ and $age_2 = \{x \in \mathbb{Z} \mid 25 \leq x \leq 30\}$. Using the previous formulae, S_{position}, S_{span}, and S_{content} can be calculated as:

$$S_{\text{position}}(age_1, age_2) = 1 - \frac{|age_{1,l} - age_{2,l}|}{U_{age}} = 1 - \frac{|23 - 25|}{120 - 0} = 1 - \frac{2}{120} = \frac{118}{120},$$

$$S_{\text{span}}(age_1, age_2) = \frac{l_{age_1} + l_{age_2}}{2 \cdot l_s} = \frac{(26 - 23) + (30 - 25)}{2 \cdot (30 - 23)} = \frac{8}{14},$$

$$S_{\text{content}}(age_1, age_2) = \frac{\text{inters}}{l_s} = \frac{26 - 25}{30 - 23} = \frac{1}{7},$$

which results in a total value similarity of:

$$S_{\text{value}}(age_1, age_2) = \frac{S_{\text{position}}(age_1, age_2) + S_{\text{span}}(age_1, age_2) + S_{\text{content}}(age_1, age_2)}{3}$$
$$= \frac{\frac{118}{120} + \frac{8}{14} + \frac{1}{7}}{3} = \frac{\frac{713}{420}}{3} = \frac{713}{1260} \approx 0.566.$$

The other quantitative attributes types, *ratio* and *absolute* values, are special cases of the quantitative *interval* type, with $\alpha_{i,1,l} = \alpha_{i,1,u}$, $\alpha_{i,2,l} = \alpha_{i,2,u}$, $l_{\alpha_{i,1}} = l_{\alpha_{i,2}} = \text{inters} = 0$.

Qualitative Attributes. As shown in [GD92, GR95], the similarity of qualitative attributes can be measured as the similarity due to *span* and *content*, since there is no meaningful measure of *position*.

Let $l_{\alpha_{i,1}}$ be the length or number of elements in $\alpha_{i,1}$, $l_{\alpha_{i,2}}$ accordingly for $\alpha_{i,2}$. Let also inters = number of elements common to $\alpha_{i,1}$ and $\alpha_{i,2}$, and l_s = combined span length of $\alpha_{i,1}$ and $\alpha_{i,2} = l_{\alpha_{k,1}} + l_{\alpha_{k,2}} - \text{inters}$.

The similarity due to *span* is then defined as:

$$S_{\text{span}}(\alpha_{i,1}, \alpha_{i,2}) = \frac{l_{\alpha_{i,1}} + l_{\alpha_{i,2}}}{2 \cdot l_s}.$$

The similarity due to *content* is defined as:

$$S_{\text{content}}(\alpha_{i,1}, \alpha_{i,2}) = \frac{\text{inters}}{l_s}.$$

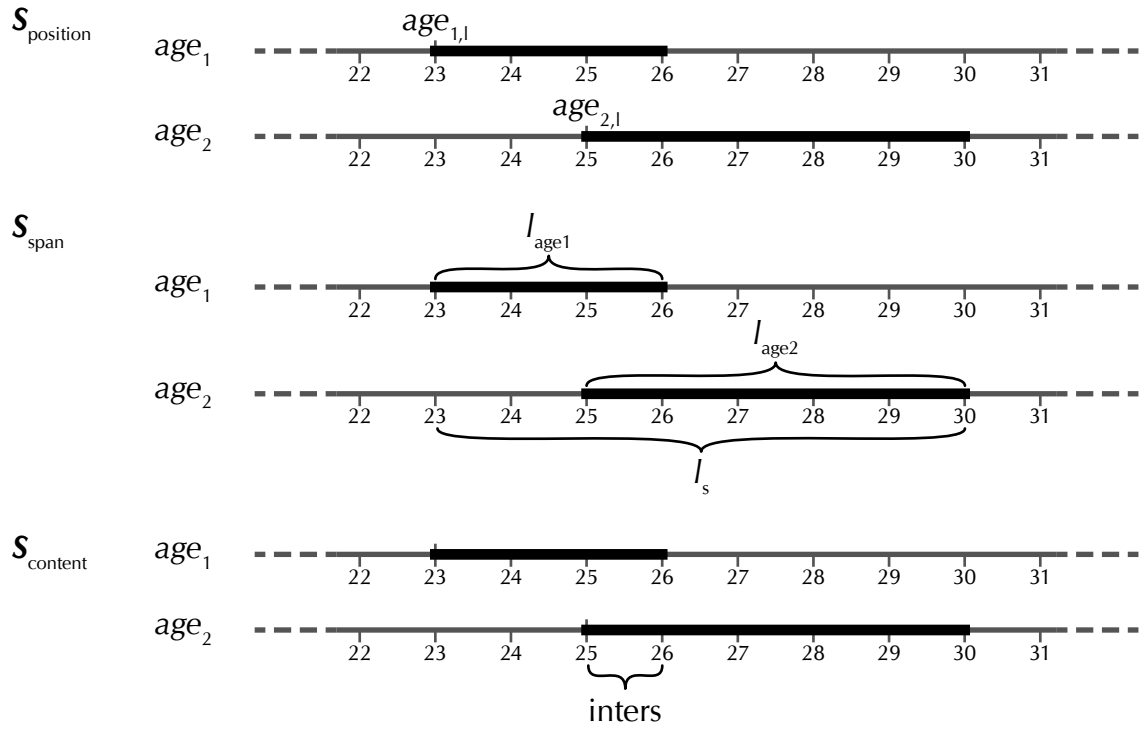


Figure 4.2: Value similarity for a quantitative interval attribute *age*

For qualitative attribute types, the total value similarity is therefore the arithmetic mean of S_{span} and S_{content}

$$S_{\text{value}}(\alpha_{i,1}, \alpha_{i,2}) = \frac{S_{\text{span}}(\alpha_{i,1}, \alpha_{i,2}) + S_{\text{content}}(\alpha_{i,1}, \alpha_{i,2})}{2}. \tag{4.4}$$

Structured Attributes. Contextcast supports structured attributes with an underlying hierarchy of values, such as a *type* hierarchy. We obtain the similarity of such values by their proximity in the hierarchy over the total size of the hierarchy. Formally, let n_1, n_2 be two nodes that correspond to the values of an attribute α_i in a given hierarchy and h the height of this hierarchy. Let $L(n)$ denote the level of a node n in this hierarchy. Then, the similarity between n_1 and n_2—and thus the value similarity for α_i—is computed using the sum of the distance of n_1 and n_2 from their first common ancestor a over twice the height h of the hierarchy:

$$\begin{aligned} S_{\text{value}}(\alpha_{i,1}, \alpha_{i,2}) &= S_p(n_1, n_2) \\ &= 1 - \frac{(L(n_1) - L(a)) + (L(n_2) - L(a))}{2 \cdot h} = 1 - \frac{L(n_1) + L(n_2) - 2L(a)}{2 \cdot h} \end{aligned} \tag{4.5}$$

Overall Value Similarity. We mentioned before that Gowda et al. simply add the value similarities of the attributes that two objects—or contexts in our case—share.

$C_1 = C_2$:	$\alpha_1 = 2$	$C_3 = C_4$:	$\alpha_4 = 12$
	$\alpha_3 = 42$		$\alpha_7 = 8$
			$\alpha_{11} = 2$

Table 4.1: Example: Value similarity of pairs of identical contexts

This leads to unintuitive results. In the case of identical contexts, for instance, this can lead to two identical contexts being "more identical" than another two, also identical contexts. Table 4.1 shows such an example of identical contexts C_1 and C_2 as well as C_3 and C_4.

Obviously, since the pairs share the same attributes, $S_{\text{structural}}(C_1, C_2) = S_{\text{structural}}(C_3, C_4) = 1$. Simply summing up the value similarities for the pairs, however, leads to unexpected results: Since the values of the attributes in the contexts are identical, the value similarity for α_1, α_3, α_4, α_7, and α_{11} equals 1. Thus, a simple addition means that $S_{\text{value}}(C_1, C_2) = 2$ and $S_{\text{value}}(C_3, C_4) = 3$. Thus, C_3 and C_4 would be considered "more similar" than C_1 and C_2, which is nonsensical for pairs of identical contexts.

Therefore, we use the arithmetic mean of the value similarities of all attributes that two contexts C_1 and C_2 share to compute the value similarity of the two.

Definition 4.5 (Value Similarity)**.** Let A_{shared} denote the attributes that C_1 and C_2 have in common: $A_{\text{shared}} = \texttt{Attributes}(C_1) \cap \texttt{Attributes}(C_2) = \{\alpha_1, ..., \alpha_k\}$ (without loss of generality). The *value similarity* S_{value} between the two contexts C_1 and C_2 can then be expressed as

$$S_{\text{value}}(C_1, C_2) = \begin{cases} \frac{\sum_{i=1}^{k} S_{\text{value}}(\alpha_{i,1}, \alpha_{i,2})}{k}, & \text{for } A_{\text{shared}} \neq \emptyset \\ 0, & \text{else} \end{cases}. \tag{4.6}$$

Combined Structural and Value Similarity

For the overall similarity of two contexts, we combine the structural and the value similarity as follows: multiply structural similarity of the two contexts with the arithmetic mean of the value similarity of all attributes that occur in both contexts. Formally, with attributes $\alpha_1, ..., \alpha_k$ occurring in both C_1 and C_2 and n attributes in the aggregated context, this leads to

$$\begin{aligned} S(C_1, C_2) &= S_{\text{structural}}(C_1, C_2) \cdot \frac{1}{k} \sum_{i=1}^{k} S_{\text{value}}(\alpha_{i,1}, \alpha_{i,2}) \\ &= \frac{k}{n} \cdot \frac{1}{k} \sum_{i=1}^{k} S_{\text{value}}(\alpha_{i,1}, \alpha_{i,2}) = \frac{1}{n} \sum_{i=1}^{k} S_{\text{value}}(\alpha_{i,1}, \alpha_{i,2})). \end{aligned} \tag{4.7}$$

This multiplication of the two components matches the intuitive notion that for contexts with identical attribute sets, thus $S_{\text{structural}}(C_1, C_2) = 1$, the result is the arithmetic mean of the value similarity. For contexts with disjoint attribute sets, and thus $S_{\text{structural}}(C_1, C_2) = 0$, the overall similarity is 0, independent of the value similarity (which is also 0, since there are no overlapping attributes).

4.4.4 Continuous Context Aggregation

Clients can register and deregister contexts at any time and the design of a context-aware communication mechanism needs to account for this. Based on the pairwise aggregation defined in Section 4.4.2 and the similarity measure from Section 4.4.3, we propose the following algorithms. These algorithms are designed to provide a continuous aggregation of new, similar contexts and a disaggregation of removed contexts to fulfill the continuous operation requirement from Section 4.2.

Context Addition. The algorithm we propose for adding a new context is similar to centroid-linkage algorithms in cluster analysis (see, e.g., [XWI05]). Using the similarity measure from the previous section, the system can compute the similarity between pairs of contexts, either singleton ones or the results from a previous aggregation. We can then aggregate a newly added context with an existing one as follows: When a ContextRouter receives a new context C_{add} from one of its neighbors, it computes the similarity between C_{add} and all other contexts in the routing table for this neighbor. If the highest similarity between C_{add} and an existing context C_{max} exceeds a configurable similarity threshold, S_{th}, the system aggregates C_{add} and C_{max}. If the similarity is not above the threshold, C_{add} is considered too different from any existing context and must therefore be added to the routing table without aggregation.

The similarity threshold S_{th} serves as a parameter for the operation of Algorithm 4.3. It determines how similar two contexts must be before they are aggregated and thus whether the routers have a very fine view of client contexts, with a high update rate, or a rather coarse view, with a low update rate. This allows to trade off context update load against false positive load. By adjusting its value, either manually by a human administrator or autonomously, one can optimize the system for the observed contexts and messages. We evaluate reasonable values for S_{th} under different scenarios in Section 4.5.

If C_{add} is aggregated with C_{max} and the resulting context C_{new} differs from C_{max}, C_{new} must be propagated as an update for C_{max}. In this case, we call C_{add} a "defining context" for C_{new}. This is important for the context removal; when a defining context is removed from an aggregation, we need to recompute the aggregation from the remaining parts. For this purpose, a router stores individual

contexts that it aggregates but only propagates their aggregation(s) to its neighbors. Algorithm 4.3 shows this sequence in detail.

Algorithm 4.3 Context Addition

Require: A newly added context C_{add}, received from neighbor N_C.
Ensure: The information of C_{add} is added to the routing table.

```
local context store ← local context store ∪ C_add        ▷ Stored for recalculation
S_max ← 0                                                ▷ The highest similarity value
C_max ← ∅                                                ▷ The most similar context
for all (C_i, N_i) ∈ routing table do
    if N_i ≠ N_C then                                    ▷ Entry from a different neighbor
        continue
    end if
    S_i ← S(C_add, C_i)
    if S_i > S_max then
        S_max ← S_i
        C_max ← C_i
    end if
end for                                  ▷ C_max now contains the most similar context
if C_max ≠ ∅ ∧ S_max > S_th then
    C_new ← aggregatePairwiseContexts(C_max, C_add)      ▷ Algorithm 4.2
    if C_new ≠ C_max then                                ▷ Adding C_add has changed C_max
        Mark C_add as defining context for C_new.
        Replace (C_max, N_C) in the routing table with (C_new, N_C).
        Propagate C_new as update for C_max to all neighbors except N_C
    end if
else
    Record (C_add, N_C) in routing table
    Propagate C_add to all neighbors except N_C
end if
```

Algorithm 4.3 fulfills the first four requirements we list in Section 4.2:

1. It reduces the rate of updates as the merging of attribute values creates value ranges instead of exact values. Thus, updates need only be propagated if an update falls outside of an aggregation, i.e., adds a new attribute or the value of an attribute is no longer covered by the range of the aggregated values. Additionally, the value ranges are derived automatically from the similarity of client contexts; otherwise, the system would quickly degenerate to use aggregated contexts that contain ranges for every value that an attribute can

take. Also, since multiple clients are represented by a single context, it also reduces the amount of contexts that is propagated to the routers.

2. The context aggregation is completely transparent for applications. An existing application can continue to register contexts with exact values, the ContextRouters aggregate similar ones when propagating the information into the network. However, the approach does not prevent updated clients from registering contexts with inexact values, for example to increase privacy by not revealing exact information about a client.

3. It preserves the existing Contextcast delivery semantics due to the adjusted evaluation of constraints for aggregated values. A message is forwarded to at least all the ContextNodes as with exact client contexts; but because of the information loss when aggregating contexts, they might be forwarded to additional nodes that do not have a matching recipient connected.

4. The application of a centroid-linkage algorithm to aggregate a new context update allows for an efficient continuous operation. In particular, it does not require that a router reanalyzes all components of existing aggregations once an update arrives. Such a continuous re-clustering (see Section 4.6) of client contexts would place an enormous load on ContextRouters due to the dynamic nature of client contexts.

Context Removal. The removal algorithm for contexts (e.g., when a client disconnects from the system) complements Algorithm 4.3. Basically, when a context C_{rem} is supposed to be removed from the routing table, a ContextRouter first checks whether C_{rem} is a singleton context or part of an aggregation. A singleton context can simply be removed from the routing table and its removal propagated to neighbors. If the context is part of an aggregation, the removal depends on whether it is a defining context for this aggregation. If it is not (i.e., its addition had no effect on the aggregation), its removal does not affect the aggregation either. If it is a defining context, the algorithm needs to recalculate the aggregation from the contexts it contains, and potentially propagate an update for the aggregation. See Algorithm 4.4 for a detailed description.

Algorithm 4.4 complements Algorithm 4.3, especially with regard to the efficient continuous operation requirement when contexts need to be removed. If a context was part of an aggregation, removing it requires only a recalculation of this particular aggregation, other aggregations are not affected, thus also fulfilling the fourth requirement.

Aggregation Degeneration. While our aggregation approach supports a continuous aggregation and disaggregation of client contexts, its results depend on the order

Algorithm 4.4 Context Removal

Require: A context C_{rem} received from N_C to remove from the routing table.
Ensure: The information of C_{rem} is removed from the routing table.

```
if C_rem is a singleton context then
    Remove C_rem from the routing table
    Forward C_rem to all neighbors except N_C
else                                   ▷ C_rem is part of an aggregation C_aggreg
    if C_rem is defining context for C_aggreg then
        Recalculate C_aggreg from local context store        ▷ Cf. Algorithm 4.3
        Propagate an update of C_aggreg to all neighbors except N_C
    else
        Do nothing
    end if
end if
local context store ← local context store \ C_rem          ▷ Remove obsolete context
```

of context additions and removals. For example, given a suitable sequence of context registrations, an aggregation may grow to represent a very generic context, i.e., a combination of many attributes with wide ranges of values. Similarly, removing contexts from an aggregation may lead to aggregations that are too generic, due to values no longer present in an attribute's value range.

In both cases, an aggregation may incorrectly match many messages, thus causing a lot of false positives in the network. This obviously runs contrary to the goal of lowering the overall network load. Such a degeneration of aggregated contexts is rather easy to detect, though.

In Chapter 5, we present a number of statistics that ContextRouters maintain for the adaptive propagation of client contexts. Among those statistics is the rate of false positive messages that each propagated context generates. We can easily extend this approach to aggregated contexts; i.e., if a router detects that a propagated, aggregated context causes too many false positive messages, this aggregation has degenerated in some form.

Once an aggregated context C_{degen} is identified as causing too many false positives, the ContextRouter can correct the situation: The router marks C_{degen} so as not to aggregate any more contexts with C_{degen}. Then treat all the contexts that make up C_{degen} as new contexts, aggregating them again if possible, but in a randomized order. This way, the individual contexts are either aggregated with other, more similar (composite) contexts or propagated as singleton contexts if they are too dissimilar from all other contexts. Finally, the neighbors that had received C_{degen} are informed that C_{degen} is no longer valid. Algorithm 4.5 details these steps.

Algorithm 4.5 Re-Aggregation

Require: A degenerated context C_{degen}.
Ensure: The information of C_{degen} is added to the routing table, possibly aggregated with other contexts.
Mark C_{degen} as deprecated ▷ No longer a candidate for aggregation
for all $C_{\text{component}} \in C_{\text{degen}}$ **do** ▷ In random order
Add $C_{\text{component}}$ as a new context ▷ Do not consider C_{degen} for aggregation
end for
Invalidate C_{degen} with all neighbors that it was propagated to

4.4.5 Optimized Aggregation Candidate Selection

For a newly added context, a node must calculate its similarity with all other contexts, aggregated or singleton contexts, that are attached to the same link. Especially when there are a large number of contexts attached to a link, this calculation can cause a rather large amount of load for the node. However, it is possible to simplify this calculation by first selecting a suitable set of aggregation candidates.

The similarity of two contexts consists of the product of the structural and the value similarity. By employing the concept of Bloom filters [Blo70], it is possible to calculate an upper bound of the structural similarity very efficiently: Let b_i be a bit string, which represents the structure of a context C_i as follows: Every attribute in C_i is hashed to a position in b_i, which is set to 1. Also, let $|b_i|$ be Hamming weight of b_i, i.e., the number of 1 bits in b_i. Then, the structural similarity between C_{existing} and C_{add} can be expressed as

$$S_{\text{structural}}(C_{\text{existing}}, C_{\text{add}}) = \frac{|A(C_{\text{existing}}) \cap A(C_{\text{add}})|}{|A(C_{\text{existing}}) \cup A(C_{\text{add}})|} \approx \frac{\left|b_{C_{\text{existing}}} \wedge b_{C_{\text{add}}}\right|}{\left|b_{C_{\text{existing}}} \vee b_{C_{\text{add}}}\right|} \tag{4.8}$$

Binary AND and OR can be computed very fast on conventional computer systems; for counting the 1 bits, efficient algorithms exist and some processors even offer a dedicated machine instruction such as POPCNT. The hashing of attribute names to bit positions can be done once and cached.

From this structural similarity, we can now derive a set of candidate contexts as follows: for a given similarity threshold S_{th}, we exclude a context from the candidate set (and thus skip the calculation of the value similarity) if the structural similarity is smaller than S_{th}: From the condition $S = S_{\text{structural}} \cdot S_{\text{value}} > S_{\text{th}}$, we can derive that also $S_{\text{structual}} > S_{\text{th}}$ (for the highest possible value similarity, $S_{\text{value}} = 1$).

However, the hashing of attribute names can lead to collisions and thus a higher structural similarity value. Thus, for two contexts with a similarity above the

threshold, the algorithm needs to recalculate the structural similarity based on the actual attribute sets instead of their bit string representations, in addition to the value similarity.

4.5 Evaluation

To determine the impact of using coarse information for Contextcast routing, we simulated both the approach based on approximated locations (see Section 4.3) as well as the general aggregation of client contexts (see Section 4.4). We compare the resulting number of updates and messages to the reference routing we presented in Section 3.3.3.

4.5.1 Coarse Location Information

Setup

To evaluate the performance of the location aggregation, which we described in Section 4.3, we have implemented a prototype to determine the savings in messages of the approach. We simulate an overlay network of 1 000 ContextRouters, connected with 2 000 links in total. The links are established according to a Waxman [Wax88] model. Two nodes u and v are connected in this model with probability

$$P(u,v) = \alpha \cdot e^{\frac{-d(u,v)}{\beta L}},$$

with $0 < \alpha \leq 1$, $0 < \beta \leq 1$, $d(u,v)$ the Euclidean distance between u and v, and L the maximum distance between any two nodes.

In this network, we simulate 1 000, 2 500, 5 000, 7 500, and 10 000 mobile clients. Since we only consider the effects of the *location* attribute, we fix the values of other context attributes and simulate only the movement of clients with these attributes. (This also means that we do not simulate any updates caused by changes in attributes other than *location*.)

The clients follow an incrementally changed random motion [HP98], a variant of a random walk mobility model; their starting points are uniformly distributed over the simulation area. We simulate clients with a low average speed typical for pedestrians. The simulated system covers a total area of 10 000 units $\times$ 10 000 units, which we then divide up into 400 squares (resulting in an edge length of 500 units per square). On average, 60 % of these are selected as access networks and each access network is assigned a router within its area as ContextNode.

Each of the clients sends random messages with an average rate of one message every 18 s; the target location of these messages are random squares with an edge

length of 0.5 to 1.5 times the edge length of an access network square (500 units). The target locations are uniformly distributed over the total simulation area. The other addressing constraints are set to the fixed attribute values, therefore only the *location* is relevant for determining matching recipients.

In this system, we then simulate our approach approximating client location by the access nodes' service areas (Service Area Approximation (SAA)) and compare it to a number of Distance-based (Reporting) (DB) approaches; with distance-based reporting, a client updates its *location* in the system once the distance from its last reported position exceeds a given threshold (cf., e.g., [Leo03, FLR07]). The DB approaches use a number of different distance thresholds, denoted as DBn, where n corresponds to the distance threshold; i.e., DB20 denotes a distance threshold of 20 units. The one end of the range, DB20, provides a higher location accuracy. This takes the typical accuracy of GPS into account, without producing unnaturally high update rates by reporting movement of 1 unit or less. The other end of the range, DB500, uses a distance threshold of 500 units. This is the same as the edge length of the service areas for the SAA, which results in a similar location uncertainty for SAA and DB500.

In all approaches the clients also send an update both when entering and leaving an access network. This also means that, if the distance threshold were significantly higher than the size of an access network, the distance-based reporting would behave practically identical to SAA: a moving client leaves a service area, which causes an update, before it reaches the distance threshold. An update is not sent during movement inside an access network, since their size is smaller than the reporting threshold. Thus, a node sends an update only when leaving one service area and entering another one.

The run time of such an experiment is limited to 900 s; thus, every client sends 50 messages on average during the experiment. We run each experiment 10 times, with different network topologies, client movement, and Contextcast messages. In each run, we measure the number of propagated updates and forwarded contextual messages and calculate the arithmetic mean and the standard deviation of the simulation runs.

Network Load Against Number of Clients

Figure 4.3 and Figure 4.4 illustrate the measured update and message load for SAA and DB250 (as one representative of the various DB approaches), respectively.

As one can see from Figure 4.3, the amount of updates is proportional to the number of clients. This is an obvious result of the approach as any additional client creates additional updates according to its movement pattern.

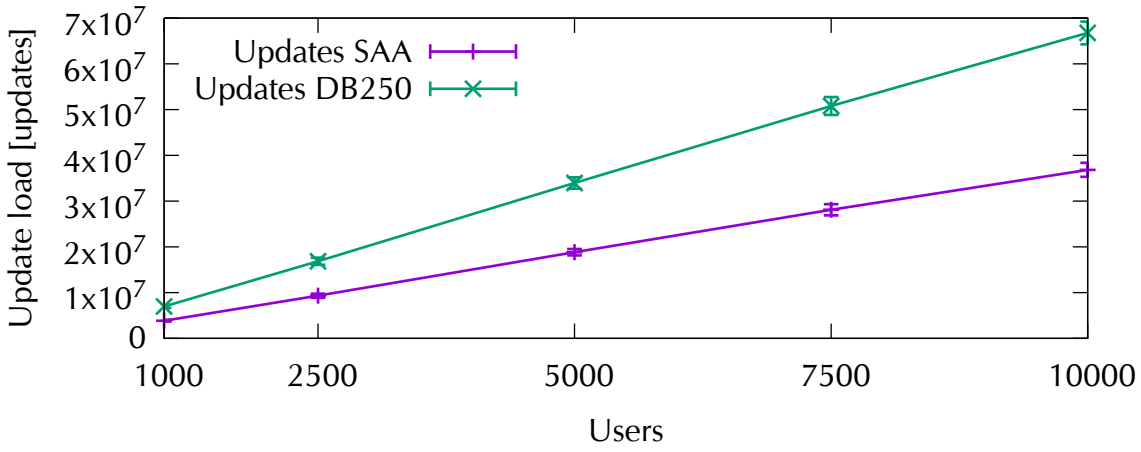


Figure 4.3: Update load against number of clients

Similarly, Figure 4.4 also shows that the message load is roughly proportional to the number of clients. The reason for this is the same as for the update load: each client sends a message at an average rate of one message every 18 s. Thus, if the number of clients doubles, so does the amount of messages sent. It is, however, not perfectly proportional. For lower number of clients, we measure less messages then one would expect. In particular, for DB250 and 1 000 clients, we observe significantly less than $\frac{1}{10}$ of the messages compared to 10 000 clients. This is due to the lower density of clients: with fewer clients, messages generally need to be forwarded to fewer access nodes since chances are higher that there is no recipient in a message's target *location*.

Also note that—in terms of absolute numbers—the scenario is clearly dominated by the update load. This is due to updates being broadcast throughout the Contextcast system, whereas message are disseminated using a directed forwarding. This limits the amount of forwarded messages toward those service areas where matching recipients exist. We show two approaches to limit this broadcast of updates in Section 4.4 and Chapter 5.

Network Load against Coarseness of *location*

As can be seen from Figure 4.5, increasing the distance threshold of the DBn approaches lowers the amount of location updates in the system significantly. DB500 shows an almost 89 % lower amount of updates in the system compared to DB20.

The number of forwarded messages for the various DBn approaches remains more or less constant, independent of the distance threshold n. This is due to the routers being unaware of the inherent uncertainty of the *location* information that

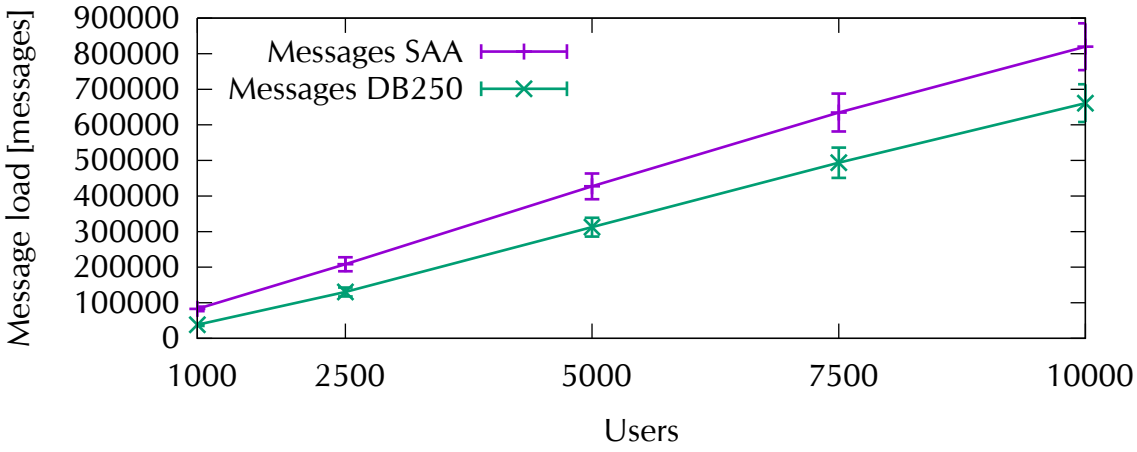


Figure 4.4: Message load against number of clients

results from the distance-based reporting. In essence, the routers treat the *location* as exact, forwarding if and only if a client's *location* is indeed inside the target *location* of a message.

SAA, in contrast, further reduces the amount of updates by 24.4 % compared to DB500 (or 91.6 % less updates when compared to DB20). The reason for this improvement is the fact that, while SAA sends an update only when leaving or entering a service area, the DB approaches sometimes send an update while they remain inside the same service area. For DB500, this happens when a client moves roughly along the diagonal of the quadratic service areas. Obviously, with lower distance thresholds, this happens more often.

At the same time, SAA increases the amount of messages that are forwarded in the system by almost 56 % compared to DB500. This is a result of the semantic changes to handle uncertain *location*s. With SAA, routers need to forward a message whenever a clients (uncertain) location overlaps with the target *location* of the message. Not all of these potential recipients are inside the target *location*, though. These additional messages place an additional load on the server, but it is required to ensure that all matching recipients indeed receive a given message.

The number of forwarded updates is significantly higher than the number of forwarded messages (in the order of 10^3 for DB20 and still 10^2 for DB500), though. This means that, overall, the reduction of updates greatly outweighs the false positives introduced by the approximated *location*.

These results have shown that the approximation of client *location* greatly reduces the amount of update messages in the system. Also, this reduction in update messages outweighs the introduction of false positives (cf. the last requirement in Section 4.2). As the message rate increases, though, the effect of false positives

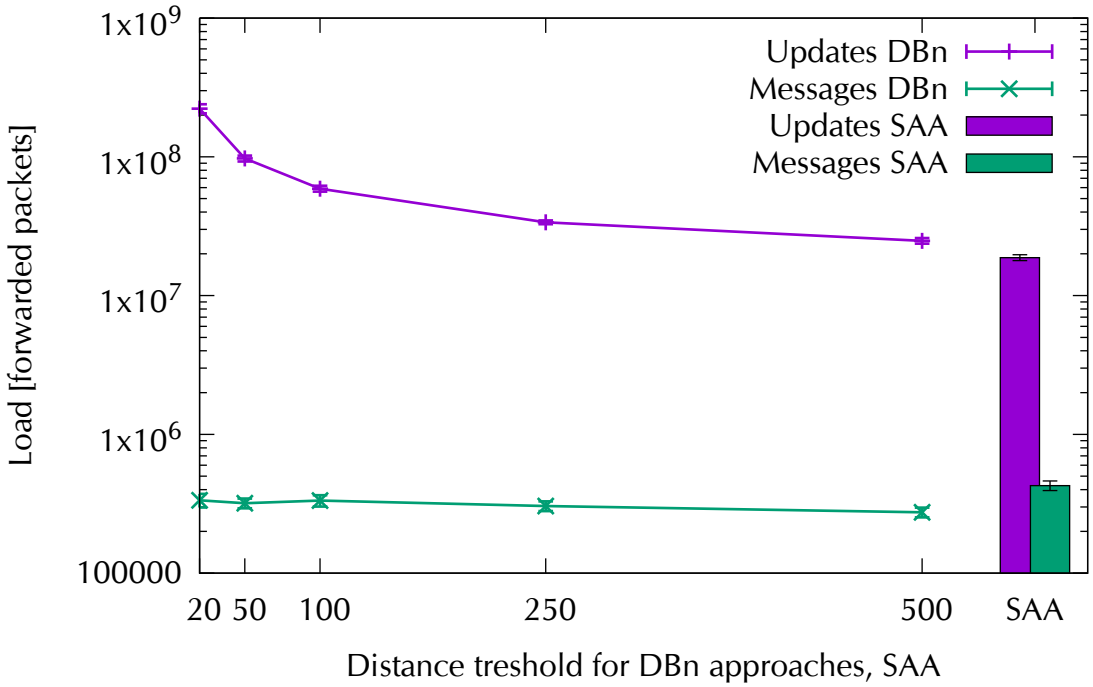


Figure 4.5: System load for DB*n* and SAA approach

would also increase until finally the false positives outweigh the savings in updates. However, the simulation shows that the message rate would have to increase by 10^2 or more to outweigh the savings achieved by location approximation. This would mean more than 5 messages every second from each client, which is unreasonably high in the scenarios we envision for a Contextcast.

Encouraged by these positive results for coarse *location*, we generalized the approach to all types of context attributes, not only *location*. In the following section, we present evaluation results of the performance impact of using aggregated client contexts for forwarding in Contextcast.

4.5.2 General Aggregation

Setup

We have implemented our method for aggregating contexts, described in Algorithm 4.2, Algorithm 4.3, and Algorithm 4.4 in a prototype implementation The simulation serves to examine the performance of the approach and to determine the influence of the *similarity threshold* parameter S_{th}.

Since our approach reduces the load in the overlay, we have chosen to implement the prototype in the PeerSim [MJ09] network simulator.

For the simulation, the Contextcast network topology consists of 400 ContextRouters, which are connected to form an undirected acyclic graph. Links are constructed according to the Heuristically Optimized Trade-off model (cf. [FKP02]): the nodes are sequentially added and uniformly distributed over an area of size $[0,1] \times [0,1]$, with the first node n_0 being the root node of the network graph. A newly added node n_i establishes a connection to an existing node n_j, which minimizes the weighted distance metric

$$f_i(j) = \gamma \cdot d_{ij} + h_j,$$

where d_{ij} is the Euclidean distance between the nodes i and j and h_j is the network distance, i.e., the number of hops, from node j to the root node n_0. The parameter γ of this weighted distance was selected as $\gamma = \sqrt{n}$ for a network of n nodes. (The authors of [FKP02] use α for this parameter, however in Contextcast α designates a context attribute.) According to [FKP02], this leads to an Internet-like power-law distribution for the node degrees.

A fraction of 60 % of these nodes are then selected as access nodes for recipients of Contextcast messages and assigned a rectangular service area. The edge lengths of these service areas are uniformly distributed between 0.04 and 0.06. Each service area is then placed such that the corresponding ContextNode is in its center. Any service area that extends beyond the $[0,1]$ interval in either direction is cut off to remain within the boundaries of the simulation.

Not all routers with degree 1 are assigned an access network, however. But, even though this means no client is ever connected to such a node, they can still be used to send messages, e.g., by corporations or NPOs. For example, a provider of commercial messages might operate its own overlay node to send its messages but no client ever connects to such a pure message source.

For the simulation, we use a simplified attribute model. It contains an attribute *location*, 3 structural attributes and 7 quantitative attributes. The hierarchy for each of the structural attributes is generated randomly with a height uniformly distributed between 2 and 5 and between 0 and 3 children per node. Each quantitative attribute has its own value range as well as an associated normal distribution with distinct means and standard deviations for each attribute.

Based on the this model, we simulate client contexts in the following way: on average, there are 2 400 clients connected to the system in total, which corresponds to 10 in each ContextNode's service area on average. Each of these contexts has an associated *location*, assigned such that it is identical to its access network's service area. With probability 0.5, a context contains a structural attribute; this is selected uniformly out of the three structural attributes in the model. Additionally, each context contains either 2 and 3 quantitative attributes, again uniformly chosen out of the 7 quantitative attributes of our model.

The value for a structural attribute is chosen by descending into the associated hierarchy, at each node uniformly choosing one of the children; the probability to further descend into the hierarchy is chosen such that it favors deep nodes. In the case of a type hierarchy, this corresponds to favoring a detailed type specification over a more general one. The values of the quantitative attributes are chosen according to the normal distribution associated with each attribute.

For the simulation, we chose three scenarios: a message-dominated one, a balanced one, and an update-dominated one. The rates of messages (r_m) and updates (r_u) in these scenarios have the ratios $r_m : r_u = 5 : 1$, $r_m : r_u = 3 : 3$, and $r_m : r_u = 1 : 5$, respectively. Updates are generated randomly, according to the following instructions: for a fraction of 70 % of all updates, the update results from a change of a single attribute. If the attribute that is being changed is the location, a node uniformly selects one of its ten nearest neighboring ContextNodes and connects to it, updating its location; for the other attributes, the values are changed in the same manner as they are created for new contexts. For the remaining 30 % of updates, a uniformly selected context is removed and a different random context—created in the previously described way—added. This models clients changing their contexts as well as leaving and new ones joining the system, respectively.

Context messages are also sent randomly and created similar to context updates: they contain constraints on location, with $p = 0.5$ on a structured attribute, and between 0 and 2 on quantitative attributes, again selected from the attribute model described before.

For our evaluation, we run five simulations with different random seeds and compute the arithmetic mean of the simulation runs, to account for random effects; unless stated otherwise, the standard deviation of these runs was negligible. To quantify the performance of our approach, we measure the amount of context updates and false positive messages.

Update Messages. As we discussed previously, aggregating client contexts conceals some of the clients' updates from the network, thus lowering the update load on routers. Therefore, we measure the number of updates that the routers actually forward to their neighbors, both with and without an aggregation of client contexts. We denote u the number of updates that are propagated in the unaggregated case and u_a the number of updates propagated with the aggregation of similar contexts performed.

False Positives. Besides the load of update messages, an aggregation of contexts causes load in the overlay network by transmitting messages without matching recipients, i.e., false positives. False positive messages can always be filtered closer to the recipient with more detailed knowledge, however. In the worst case, this may only be possible at the access node or on a client's mobile device. We therefore measure the amount of false positive messages, fp, caused by the loss of routing

information when aggregating contexts. N.b., the reference Contextcast routing does not forward false positive messages.

Update Load Reduction: Impact of Similarity Threshold

Similarly to coarse *location* information, we evaluate the impact of an aggregation of client contexts by comparing the amount of false positives caused by the aggregation to the reduction in updates. We carry out simulations with similarity thresholds S_{th} of 0.5, 0.6, and 0.7. These similarity thresholds allow an administrator to adjust the granularity of the aggregation from coarse routing information to more detailed, respectively.

Each experiment covers a time of 3 000 simulation cycles; however, for the results, we limit the simulation time to 2 500 cycles. The first 500 cycles of the simulation contain a warm-up phase, during which the contexts are first registered with the system; this generates a significant amount of updates, however, no messages are sent during that time. Limiting the evaluation to the cycles 500 to 3 000 eliminates the warm-up phase from the measurements.

Figure 4.6, Figure 4.7, and Figure 4.8 show the results for the update-dominated, the balanced, and the message-dominated scenario, respectively.

As was to be expected, forwarding with aggregated context information causes a certain amount of false positives. The figures show that lowering the similarity threshold increases the number of false positives: going from $S_{\text{th}} = 0.7$ to 0.6 increase the amount of false positives by 337.4 % in the update-dominated scenario, by 253.1 % in the balanced scenario, and by 222.2 % in the message-dominated scenario. Lowering it from 0.6 to 0.5 causes only a further increase by 36.1 %, 57.4 % and 70.4 %, respectively.

The smaller increase when lowering S_{th} from 0.6 to 0.5 can easily be explained: there is an upper limit on the amount of false positives for any given message, no matter how coarse the aggregation of the contexts actually is. The upper limit is a result of the target location of every message. As soon as a message is delivered to all access networks whose service area intersects a message's target location the delivery is complete. Thus, the maximum number of false positives any message can cause results simply from delivering it to all access networks that overlap the target location.

Overall, the increase in false positives fits the intuition that a coarser aggregation provides ContextRouters with a fuzzier knowledge about client contexts. Therefore, the dissemination trees cannot be pruned as early as with complete, unaggregated knowledge on each router. As such, messages get forwarded to routers with better, less aggregated knowledge or even to the access networks before it can be determined that they have no matching recipient.

Looking at the decrease in updates, i.e., the difference between the number of updates without and with aggregation, we can see similar results in all three scenarios. Aggregating contexts causes a clear reduction of update load in the system. A similarity threshold $S_{\text{th}} = 0.6$ saves between 4.5 and almost 9.8 times the amount of updates than aggregating with $S_{\text{th}} = 0.7$. However, lowering the threshold further to 0.5, we observe that this effect reverses. If the system aggregates contexts too aggressively, the update load increases again, almost halving the savings that were achieved with $S_{\text{th}} = 0.6$. The reason for this effect is that a similarity threshold that is too low causes rather different contexts to be aggregated. When such dissimilar contexts get aggregated, there is a high probability that a context that is added to another one is actually a defining context for the resulting aggregation. Thus, another update needs to be propagated further into the network, causing more update load instead of reducing it.

When we compare the numbers for false positives and updates, we can see that by aggregating similar contexts, the amount of saved updates vastly outweighs the amount of false positives it introduces. In our simulated scenario, the amount of additional false positives is less than 0.1 % of the reduction of updates our approach achieves. In other words, only a system with much more contextual messages or much less updates due to quasi-static client contexts (either one by a factor in the order of 10^3) would not benefit from an aggregation of client contexts.

Compared to the reference dissemination (compare Section 3.3.3), our approach lowers the system load by between 18 % and almost 25 % on average for $S_{\text{th}} = 0.6$, with some scenarios achieving a reduction up to almost 30 %. Thus, the approach fulfills the last requirement from Section 4.2, a reduction of overall system load.

Also, these numbers were observed for the random contexts of the simulation. In a live Contextcast system, we expect clients that are close to each other to be more similar to one another. This allows an aggregation of contexts to work more efficiently, thus improving these results further.

Aggregation Quality over Time

Due to the dynamic nature of client contexts, new contexts are continuously registered, some existing ones removed, and others changed. This, in turn, also means that contexts are added to and removed from aggregations all the time. This might lead to a degeneration of the aggregations over time and thus a gradual increase in the number of false positives. (However, since the aggregations are becoming more general, this effect would also reduce update messages further.) To investigate this behavior, we simulate the three scenarios from before with a similarity threshold $s_{\text{th}} = 0.5$. This low threshold value leads to many contexts being aggregated. This

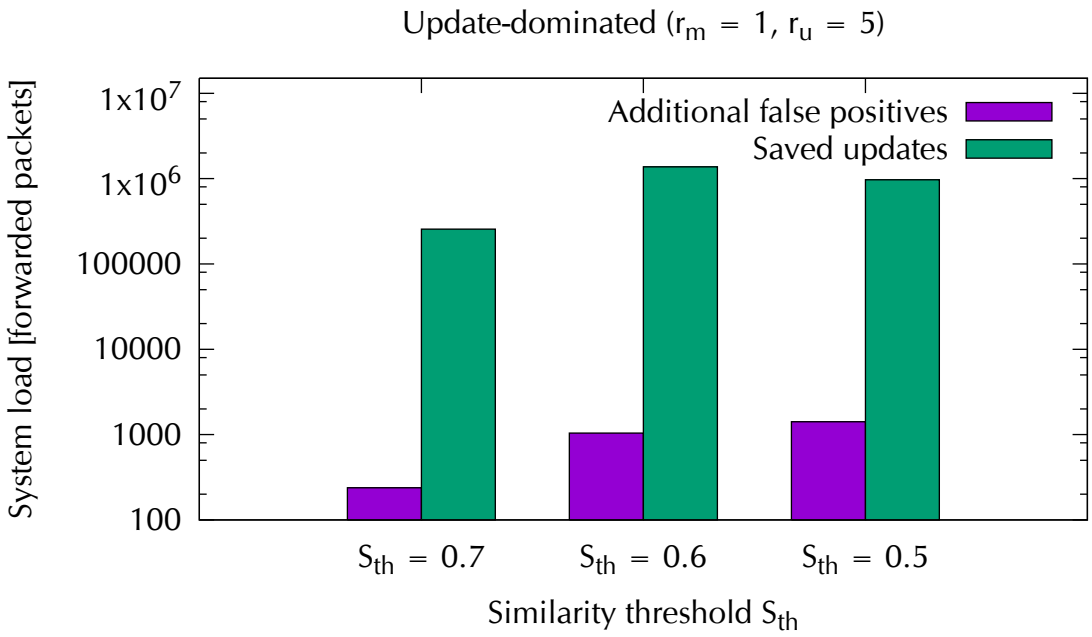


Figure 4.6: Effect of S_{th} in the update-dominated scenario

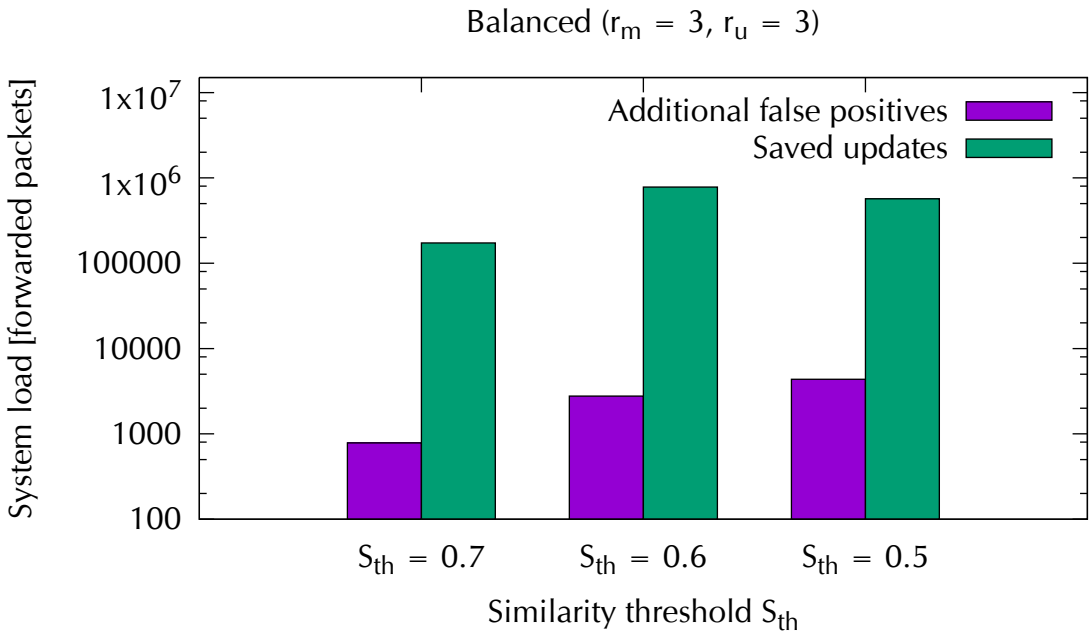


Figure 4.7: Effect of S_{th} in the balanced scenario

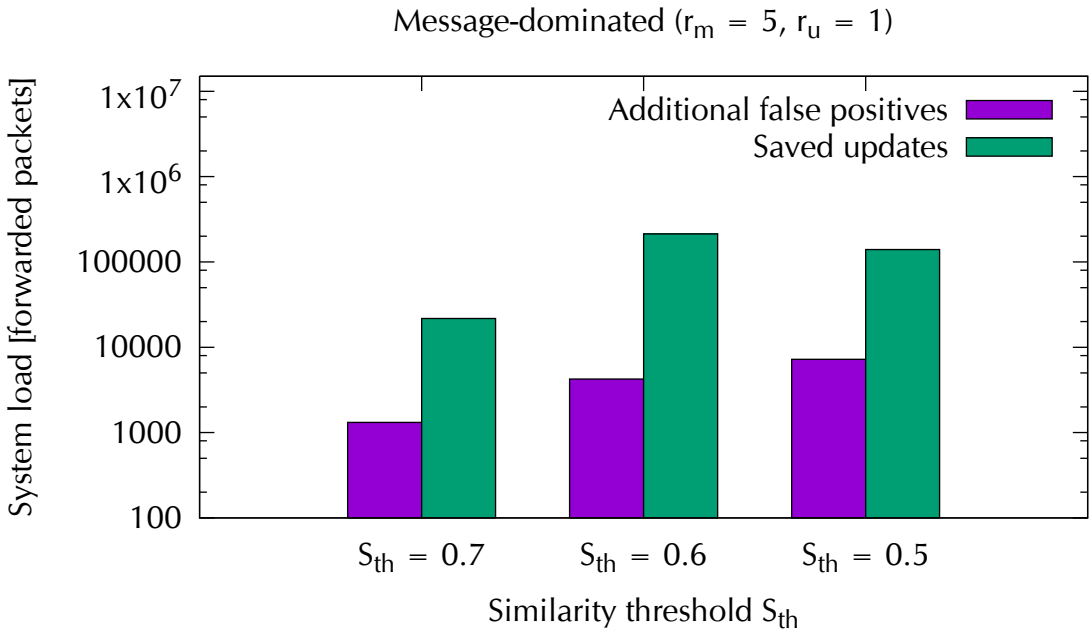


Figure 4.8: Effect of S_{th} in the message-dominated scenario

also results in a large number of changes to aggregations over time and thus should show any degeneration effect on context aggregations rather clearly.

Figure 4.9 shows the false positives during the simulation run time. To better indicate a general trend and reduce small spikes, the false positives during each simulation cycle were smoothed using a Simple Moving Average (SMA) over 25 cycles. For all scenarios, the experiment shows a time with no false positives in the beginning, the length of which differs between the scenarios. This is caused by the warm-up phases in the beginning. Their length is determined by the time it takes to register the given number of contexts with the context update rate given for each experiment. During this warm-up, no messages are sent and thus no false positives can occur.

In our experiments, the average of false positives stays relatively constant for all three scenarios during the remaining simulation time. No noticeable increase in false positives occurs over time. Therefore, even after 3 000 cycles, i.e., after about 15 000 updates in the scenario with the highest update rate, there is no discernible negative effect on the quality of the aggregations in our system.

However, it is possible that a larger change in client contexts or a particular sequence of context updates still lead to a degeneration of one or more aggregations. For example, this could happen if successive context updates cover an ever larger

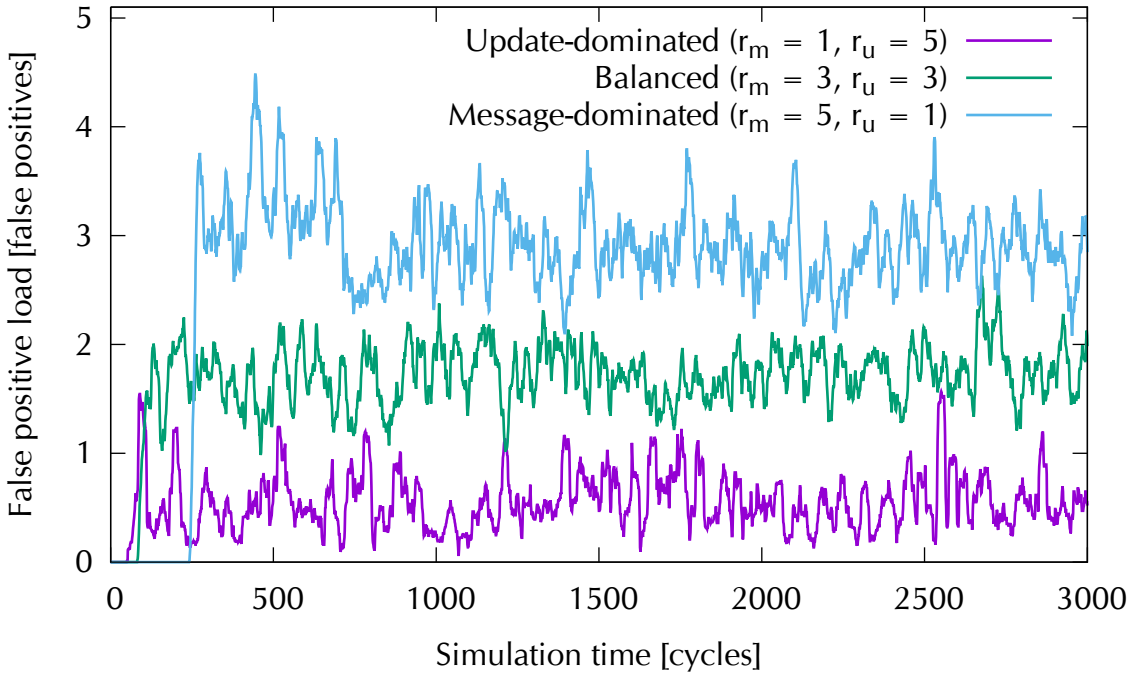


Figure 4.9: Degeneration of aggregations over time

part of an attribute's value range. This way, aggregating such updates leads to a context aggregation that is continually increasing in size until, finally, it matches practically every message.

A ContextRouter can, however, remedy such a degeneration easily: With the technique we present in Section 5.3.2, a router detects the number of false positives caused by contexts that it has propagated. When an aggregation causes many false positives, the router that created the aggregation can dissolve it and re-aggregate its partial contexts; each router keeps the partial contexts it aggregates to handle context removal. To prevent the same degeneration, the router needs to randomize the order, in which these partial contexts are then re-aggregated. As a result, some of the partial contexts may then be added to different, more suitable aggregations, while others may form new, smaller ones, which describe the actual client contexts more precisely.

Analysis

Recall, the goal of an aggregation of contexts, which we introduced in this chapter, is twofold. First, instead of m individual contexts that need to be propagated in the system, coarser aggregated contexts are propagated and used for disseminating

Contextcast messages. Second, this coarser information also reduces the update rate $u\frac{1}{s}$

Our approach is adaptive with regard to the similarity of the actual contexts in the system and how well they can be aggregated, i.e., the similarity threshold. If we assume that all clients in the system can be classified into one of k aggregations, the total number of client contexts that need to be propagated in the system then becomes $\frac{1}{k}m$.

Additionally, let us assume that each client update causes an update to one of the aggregations to be propagated throughout the network with a probability p_u. Thus, the new update rate in the system becomes $p_u u\frac{1}{s}$.

Since all updates are in a worst-case still broadcast to all n nodes in the network, we can estimate a new average update rate, depending on k and p_u as:

$$\text{Average update rate} = n\frac{1}{k}mp_u u\frac{1}{s}$$

Besides the update rate, the aggregation of contexts also causes a certain amount of false positives. We assume that each node receives an average rate of false positives of fp $\frac{1}{s}$. This leads to an additional false positive load in the network of:

$$\text{Average false positive rate} = n\,\text{fp}\,\frac{1}{s}$$

4.6 Related Work

Since context-based routing is similar to content-based routing, it seems natural to apply concepts from the field of Pub/sub systems in Contextcast. As we mentioned in Section 2.3, the author of [Müh02] describes various routing strategies, which aim to lower the load of the broker network caused by the subscription propagation. Routing schemes based on "covering" or "merging" (or both) of client subscriptions have become fairly common in content-based Pub/sub systems.

Both approaches are based on the idea that a broker does not need to propagate a more precise subscription if a more general one was already propagated. The more general subscription will match the same events (and more) that the more precise one matches. Thus the node receives a superset of events that it had received from propagating the more precise subscription.

A "Routing based on Filter Covering" relies on subscriptions "covering" other ones. Formally, this is defined as one subscription matching a superset of events of another one, thus fully covering the more precise subscription. In terms of subscriptions, this is the case, for instance, when subscriptions specify attribute constraints with values within a range of values. Let $a, b, c, d \in \mathbb{R}$, such that $a \leq b \leq c \leq d$. Then a

predicate requesting events with values $\in [b, c]$ matches a subset of events of one requesting values $\in [a, d]$, simply because $[b, c] \subseteq [a, d]$.

Client contexts in Contextcast, however, specify exact values for their context attributes. They are, in other words, point values in a context space, defined by their attributes and values. Two contexts are therefore either identical (which is actually one, albeit rare, form of covering) or differ in one or more attributes. In the latter case, neither context represents the other ones values, simply due to the nature of exact values for context attributes.

For one context attribute, *location*, the impact of value ranges, i.e., inexact locations, has been well studied. Especially the tracking of moving objects such as in MODs benefits from allowing an amount of uncertainty for the position of mobile objects. Reporting protocols allow clients to reduce their update rates while still maintaining well-defined semantics for inexact position information. Examples of such reporting protocols include distance-based reporting [Leo03, FLR07] or Dead Reckoning (DR) [WSCY99, ČJP05].

Distance-based reporting periodically compares a client's actual position $\vec{a}(t')$ at time t' with the last reported position $\vec{p}(t)$ at a time $t < t'$. Once the distance $d = |\vec{a}(t') - \vec{p}(t)|$ exceeds a threshold δ, the client reports a new position to the server. When a client approaches the limit δ, it must increase the sensing frequency of its location systems, to accurately detect the object leaving the uncertainty area. The authors of [FLR07] extend this approach with an early reporting, which trades off the energy needed for an early update against the energy needed for continuous high-frequency position sensing while d is close to δ.

Dead Reckoning transmits a user's current location together with a prediction function for future positions. The server can then compute a client's position $\vec{p}(t)$ at time t from the prediction. The client can also determine its actual position $\vec{a}(t)$ at time t with one of its location systems, e.g., GPS. If the deviation $d(t) = |\vec{a}(t) - \vec{p}(t)|$ exceeds some specified upper bound δ, the client sends an updated location and prediction function to the server.

Thus, such reporting protocols guarantee that an object's position as reported by the MOD never deviates more than δ from the actual position. Applications relying on position information can therefore—and must—adjust their operation to handle this uncertainty.

Inexact attribute values with a guaranteed range such as from reporting protocols allow us to extend Contextcast with a method similar to "cover". While this works well for the *location* attribute, other attributes are more difficult. Particularly the point-like nature of client contexts requires a more flexible approach similar to "filter merging", or "merging" for short. "Merging" allows a broker in Pub/sub to replace a number of subscriptions with a single, newly created one. Formally, this merged

subscription covers all of the initial subscriptions, i.e., it matches a superset of events of the combined individual subscriptions.

Applying this idea to Contextcast allows us to "merge" several similar point-like client contexts by describing a superset of values of the context attributes. For attributes whose values form a partially ordered set this can be the range of values of that attribute. However, it also needs to take other types of data into account. For instance, a *type* attribute based on a class hierarchy, which is important in our system, needs to be handled differently than, for instance, a quantitative attribute such as *age*.

Merging contexts in this fashion requires a reasonable selection of similar contexts for aggregation. This is closely related to the clustering of data items (cf. [XWI05] for an overview). However, existing clustering approaches can not be simply transferred to Contextcast, as it has a couple of special requirements: First, clustering algorithms with a predefined number k of clusters, such as the k-Means algorithm [Mac67], are not useful for Contextcast. Since the optimum number k of clusters is not known a priori, such an approach would require a method to determine a reasonable number k. Obviously, repeating the calculation for different values of k and then determining an appropriate value places unnecessary load on the nodes. Second, clients can join and leave the Contextcast system at any time. This eliminates those clustering algorithms that require random access to the data being clustered, e.g., the ISODATA algorithm [BH67]. Similarly, algorithms that use a random sampling of the data such as CLARA [KR05] cannot reliably determine a representative sampling of the data due to continuous addition and removal of contexts.

A continuous addition of new elements is possible with algorithms such as BIRCH [ZRL96] or more specialized stream clustering algorithms (e.g., STREAM [OMM+02]). Neither can, however, remove old elements from the data. CluStream [AHWY03] exploits a *subtractive* property of the data to generate a clustering for an arbitrary time window. However, such a subtractive property does not hold for the client contexts in our system.

Also, many clustering algorithms are tailored for objects in $\mathbb{R}^n$ or at least for objects with an explicit metric or distance function. They cannot be adapted easily for client contexts with their various different attributes, which may not provide a natural notion of distance or similarity. In particular, such measures, if they are available, are not easily comparable between different attributes. For instance, how does a distance of 500 m of *location* compare to a distance of 10 a of *age*? Gowda et al. [GD92] have researched clustering of "symbolic objects", i.e., objects represented by a set of attributes, with potentially different types. These symbolic objects are therefore very similar to client contexts in Contextcast. Based on their results we have derived a similarity measure for our contexts in Section 4.4.

4.7 Summary

In this chapter, we showed a technique to improve the scalability of Contextcast by propagating coarser context information. This reduces the load caused by context updates since only updates that fall outside of a coarse value need to be propagated. The presented approach ensures that the semantics of Contextcast is maintained by forwarding a message to at least the same set of recipients as the reference algorithm we presented in Chapter 3.

A specific context attribute, the client's *location*, can be easily approximated by the service area that the client is currently in. As long as the client moves within this service area, their *location* does not need to be updated on the ContextRouters. Since routers no longer have the exact *location* of clients, routers cannot determine whether a context satisfies a given *location* constraint. Therefore, a message needs to be delivered if a client's coarse *location* overlaps with the target *location* of the message; the client's exact *location* may be inside the overlapping area.

For arbitrary context attributes, such a discretization into coarser values is not readily available. We have shown an approach that propagates aggregated, less precise context information. If contexts are aggregated, the number of context updates that need to be propagated is lowered. Instead of several individual contexts, an aggregated context is propagated. The aggregation automatically determines an appropriate discretization of the various attributes' value ranges.

Aggregating context information in this manner, however, causes a loss of information. Previously separate attributes occur together in an aggregated context and attribute values become coarser. To minimize the amount of information loss, routers aggregate only similar contexts. The similarity of contexts is measured by two components: the structural similarity, which measures how many attributes two contexts have in common; and the value similarity, which measures the similarity of the values for an attribute that two contexts have in common. A similarity threshold allows an administrator to configure how aggressively the algorithm aggregates context information.

While aggregating similar contexts minimizes the information loss, it cannot completely eliminate it. Thus, the imprecision of aggregated contexts needs to be considered in the forwarding decision: when evaluating constraints against aggregated context information, routers forward a message if there is a potential recipient—according to the aggregated information—that can be reached via a link. This leads to an increase in message load since routers forward messages even though a router later in the forwarding process may determine (using more precise information) that no recipient actually matches the addressing. This false positive load is necessary, however, to ensure that no matching recipient misses a message due to the imprecisions introduced in the aggregation.

In our evaluation of these approaches, we found that the approximated *location* lowered the amount of update messages by more than 24 % compared to a distance based-reporting with a similar threshold. At the same time, while it increases the message load in the system, the effect is not very pronounced for the update and message rates that occur in a Contextcast system. The more general aggregation of contexts that, which applies to all attribute types, reduces the overall system load by between 18 % and 25 %, with some scenarios achieving a reduction of up to 30 %. This result is largely due to a massive decrease in update load, while the resulting increase in false positive load has only a small impact for the message and update rates in our system.

Chapter 5

Directed Forwarding using Adaptively Propagated Client Context Information

> Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.
>
> *(Antoine de Saint-Exupéry)*

5.1 Overview

In the previous chapter, we discussed an approach to reduce the update load in the Contextcast system by combining the information of similar clients and making the information coarser in the process. The approach takes the similarity of client contexts into account, thus lowering both the number m of contexts to propagate as well as the update rate for these contexts.

Despite the savings shown in Section 4.5, the system still maintains complete context information, i.e., every context update is still propagated to all n Context-Routers in the system in some form (possibly aggregated, though). This complete information and the necessary updates limit the scalability of the system. Without this context knowledge, a directed routing of contextual messages would not be possible, though: messages would have to be broadcast to reach every matching recipient. This shifts the scalability bottleneck from the update load to a message broadcast load, which does not scale, either.

However, since every context is propagated to every router, this means that a context may also be propagated to a number of routers that never require the information for a forwarding decision. For example, if context C_1 in Figure 5.1 never matches any message that is sent at routers R_2 or R_3, it is not necessary to propagate the information of C_1 from node R_1 towards R_2 and R_3. Even if there is occasionally a message $M : C_1 \sqsubset M$, maintaining the information on R_2 and R_3 may cause more

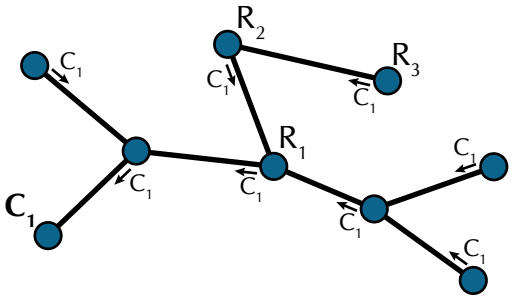


Figure 5.1: Routing table entries

updates than simply flooding M towards R_1 when it occurs. Depending on the rates of updates for a context and of such messages, this may even be the case if M turns out a false positive and is discarded at R_1.

In this chapter, we therefore present an adaptive propagation approach. It has the same goal of reducing the update load in Contextcast, however, it is orthogonal to the aggregation of client contexts we discussed in Chapter 4. An adaptive propagation takes both the update load of a piece of context information and the false positive load into account to decide whether to propagate this information to a neighboring router. The rationale behind it is to limit the propagation of client contexts only towards sources that send out messages that the propagated contexts may match; and also only if they do so at a high enough rate that maintaining context information is actually beneficial to the system. I.e., the message rate must be higher than the corresponding context information updates. This enables routers on the paths between the source and recipients to employ context information for a directed forwarding, thus pruning the dissemination tree. Routers that do not have the necessary knowledge to evaluate the constraints of a particular message fall back to speculative forwarding, i.e., they assume the presence of a matching recipient. Such a speculatively forwarded message is a potential false positive if there is no actual recipient matching the addressing.

In this chapter, we first discuss the requirements for such an approach in Section 5.2. Then, we introduce the detailed approach in Section 5.3. It builds on a monitoring of the system to determine both the load caused by context updates as well as by messages. This information is then used to determine whether propagating a client context is beneficial to the system. After that, we present the results of an evaluation we performed to show its effectiveness in Section 5.4. Finally, we discuss related approaches in Section 5.5, before closing with a short summary in Section 5.6.

5.2 Requirements

The general goal of using an adaptive propagation is the same as for using coarse information: It aims to improve the scalability of the system by reducing the amount of updates that are propagated to the routers; of course, just like in the previous chapter, the approach must not alter the Contextcast semantics. Due to the similar goals, the requirements are also very similar to the previous chapter (for details, please refer to Section 4.2): (1) Reduction of Updates, (2) Implementation Transparency, (3) Preserving Delivery Semantics, and (4) Continuous Operation. However, there is an additional requirement, Distributed Operation.

Distributed Operation. The Contextcast system consists of a network of interconnected routers, which independently forward messages from a source to all ContextNodes with matching recipients. Adaptively propagating client contexts towards sources whose messages they may match exploits local differences in messages. The decision which contexts to propagate and which not therefore needs to be a localized one. It must not depend on a global view of the system. While collecting such a state is possible, it would introduce another performance bottleneck into the system. This would run contrary to the benefit achieved by the adaptive propagation approach.

5.3 Adaptive Propagation of Client Contexts

As we discussed previously, in Contextcast we can observe two types of load on the overlay links: *message load*, which results from the dissemination of messages, and *context update load*, which is caused by the propagation of context information to routers. Message load can be further divided into legitimate message load (those messages for which a recipient exists) and *false positive* message load. The legitimate message load is required if the system is to retain its previous delivery semantics; otherwise clients with a matching context would not receive messages.

False positive load, however, is generated whenever the system forwards a message for which no matching recipient exists. As we discussed in Chapter 4, this happens when a node has imprecise information, e.g., due to coarse or otherwise incomplete knowledge about contexts, and must assume the presence of a matching recipient in the direction of a link to preserve the delivery semantics of Contextcast. We call such a message that is forwarded due to a presumed matching recipient a *speculatively forwarded* message.

Therefore, while it is possible to reduce update load by not propagating context knowledge, such an approach requires a certain amount of speculatively forwarded messages. This, in turn, also increases the amount of false positives, since not

every speculatively forwarded message finds a matching recipient in the end. Thus, the goal is to minimize the overall system load, i.e., from Contextcast messages—particularly false positives—and context updates.

Obviously, this depends on the actual messages and client contexts that occur in the system: Propagating context information is only useful for attributes/attribute combinations that actually occur in constraints. At the same time, we need to consider the dynamic of contexts, which requires updates to keep propagated information current. Our approach therefore monitors the routing and adapts it to the actual messages and client contexts in the system. The general idea is to propagate context information if and only if it reduces the overall combined message and update load. In other words, the reduction in false positives must outweigh the additional update load for the context information.

However, since the system no longer propagates each context, the ContextRouters need to be able to make their routing decision based on incomplete context information. To this end, every router must be able to decide when it has the necessary information for a directed forwarding: If a router's knowledge is sufficient, it evaluates a message's constraints for a directed forwarding; otherwise, it falls back to speculative forwarding.

In the following sections, we present the necessary changes to handle such incomplete knowledge, statistics to estimate and measure the load of context updates and false positives, as well as the algorithms that use these statistics to adaptively forward context information when it benefits overall load.

5.3.1 Incomplete Context Knowledge

As we have shown in Section 3.3.3, a directed forwarding decision is straightforward with global context knowledge available to ContextRouters: A router forwards a message over a link if and only if it knows of a matching recipient in that direction. However, for our adaptive context propagation, the system must be adjusted to maintain its semantics without this global knowledge. The following simple example illustrates the challenges that occur when routing messages without global context knowledge.

Figure 5.2 shows a small system of two ContextRouters R_1 and R_2. R_1 is also a ContextNode and within its service area are contexts C_1, C_2, C_3, and C_4. R_1 forwards only two of these contexts, C_2 and C_4, to node R_2, where this information is entered into the routing table together with R_1 as the next hop. If R_2 now evaluates a message that would match only C_1, it does not forward it since R_2 does not know about C_1. Without additional care, this would either require falling back to broadcasting every message since routers would have to assume the presence of a matching recipient or clients would not receive messages even though their context

Figure 5.2: Forwarding with incomplete information about client contexts

matches it or clients would not receive messages even though their context matches it.

To prevent this, every router must be able to decide whether it has sufficient knowledge to evaluate a given message's constraints or needs to fall back to speculative forwarding. We enable this in Contextcast by introducing the concepts of *(complete) composite contexts*.

Definition 5.1 (Composite context)**.** A *composite context* C_{comp} contains context information for a certain set of context attributes $\mathcal{A}_p = \{\alpha_i : i \in \{1, \ldots, k\}\}$ that are selected for propagation to a neighbor N.

Definition 5.2 (Complete composite context)**.** Let M be an arbitrary message that has constraints on a subset of the attributes in $\mathcal{A}_p$ and C_R a context known to R. A *complete composite context* contains a router R's *complete context knowledge* for a set of context attributes $\mathcal{A}_p$ if the following holds true:

$$\exists C_R : C_R \sqsubset M \rightarrow C_{\text{comp}} \sqsubset M \tag{5.1}$$

In other words, another ContextRouter that has received a complete composite context C_{comp} for $\mathcal{A}_p$ can evaluate all messages with constraints on the attributes in $\mathcal{A}_p$ or a subset thereof against C_{comp} instead of against individual client contexts. If there is no such complete composite context against which to evaluate a message, it can not use a directed forwarding and instead needs to fall back to speculative forwarding.

Obviously, to ensure that routers can use this information for a directed forwarding, only complete composite contexts must be propagated between routers, since only these provide the complete knowledge for a given attribute set. Regular client contexts can no longer be used for a directed forwarding, since there might always be another matching context that was simply not propagated. This in turn would mean falling back to speculative forwarding.

A router that has propagated a complete composite context C_{comp} based on a selected attribute set $\mathcal{A}_p$ is responsible that C_{comp} contains all its knowledge for $\mathcal{A}_p$ and that C_{comp} is updated when necessary, i.e., when its own context knowledge

changes from updates it receives. (We discuss the actual attribute selection and how it is based on observed messages and client contexts in Section 5.3.2.)

With this definition, we can examine the example from Figure 5.2 again. Instead of individual contexts C_2 and C_4, R_1 would have propagated a complete composite context C_{comp} for an attribute set $\mathcal{A}_p$. Based on $\mathcal{A}_p$ and the constraints in a message, R_2 can decide for every message whether to evaluate it against C_{comp} or whether to speculatively forward it.

Since a complete composite context represents a node's complete context information for an attribute set $\mathcal{A}_p$, it typically contains information of several clients as well as other composite contexts. Generally, it is constructed from all contexts that contain information on one or more attributes that are elements of $\mathcal{A}_p$. Algorithm 5.1 shows the sequence to create such a composite context for an attribute set $\mathcal{A}_p$ from all known contexts of a node. First, the node calls the function CREATEPARTIALCONTEXT (shown in Algorithm 5.2) for each locally registered context, which creates a partial context by removing all attributes that are not in $\mathcal{A}_p$. If the result is not an empty context, it is merged into the composite context for $\mathcal{A}_p$. Second, it finds all composite contexts in its routing table that contain information about one or more attributes of $\mathcal{A}_p$. For each of these, it then iterates over the partial contexts it contains and again calls CREATEPARTIALCONTEXT for each one. Again, if this results in a non-empty context, it is merged into the composite context for $\mathcal{A}_p$.

The merging of partial contexts into a composite contexts can be a simple list of the partial contexts or a more sophisticated scheme, such as an aggregation of the partial contexts. The only requirement is that the scheme fulfills Definition 4.3, the aggregation condition. This ensures that a message that matches any of the partial contexts also matches the resulting composite context. This is obviously the case for a simple list (since all partial contexts are available verbatim) and also for an aggregation as described in Chapter 4. In our prototypical evaluation (see Section 5.4), we have implemented a simple list of partial contexts.

5.3.2 System Load Statistics

In this section, we present a number of statistics, which we use to determine attribute sets $\mathcal{A}_p$ whose propagation is beneficial to the system, i.e., that reduce false positives resulting from speculative forwarding. Additionally, the statistics enable routers to identify composite contexts that are no longer useful because messages in the system have changed sufficiently. In this case, the load to update the contexts is higher than simply forwarding a message speculatively when it occurs. Such a context needs to be invalidated to stop the updates it causes, thus again lowering overall system load.

Algorithm 5.1 Composite Context Creation

Require: An attribute set $\mathcal{A}_p$ selected for propagation to neighbor N_p.
Ensure: A composite context C_{comp} for $\mathcal{A}_p$.

```
C_comp ← ∅
for all C_local ∈ {C : C was registered locally} do
    C_local,partial ← createPartialContext(C_local, A_p)
    Append C_local,partial to C_comp                ▷ Does nothing if C_local,partial = ∅
end for
for all (C_i, N_j) ∈ Routingtable do
    if N_j = N_p then                               ▷ Entry originally received from N_p
        continue
    end if
    if ∃α ∈ C_i : α ∈ A_p then                      ▷ Contains one or more attributes from A_p
        for all C_partial ∈ C_i do                  ▷ C_partial = C_i if C_i is a singleton context
            C_partial,new ← createPartialContext(C_partial, A_p)
            Append C_partial,new to C_comp          ▷ Does nothing if C_partial,new = ∅
        end for
    end if
end for
```

Algorithm 5.2 Partial Context Creation

Require: A (non-composite) context C and an attribute set $\mathcal{A}_p$.
Ensure: A composite context C_{partial} with only attributes from $\mathcal{A}_p$.

```
function createPartialContext(C, A_p)
    C_partial ← ∅
    for all α ∈ C do
        if α ∈ A_p then
            C_partial ← C_partial ∪ α
        end if
    end for
    return C_partial
end function
```

All the measurements we introduce here are taken during time windows of length t_w. The purpose of this time window is to ensure that a sufficient number of messages and updates is observed, to reduce the effect of minor fluctuations. An administrator can and should adjust the length of the time window depending on the rate of observed events.

For now, these measurements may seem rather abstract. We are showing in Section 5.3.3 how they are used to select attribute sets for propagation and invalidate older, no longer useful context information.

False Positive Rate

The *false positive rate* fp is a measure of how many false positives arrive at a node of a certain type. It corresponds to the amount of false positive load that could be avoided in the system by propagating context information.

Whenever a router receives a Contextcast message M, with constraints $\{\phi_1, \ldots, \phi_k\}$ (corresponding to attributes $\{\alpha_1, \ldots, \alpha_k\}$), one of two things can happen:

(1) The router knows another entity to which it needs to send M. This can be either a neighboring router or a matching recipient in its own access network.

(2) The router does not need to forward the message to a neighbor and knows no local recipient.

In case (1), M is a legitimate message since the router either has a matching recipient in its access network or knows about one that can be reached via a neighboring router. In case (2), M is a false positive message, which could have been avoided had the last hop router had the necessary context information to prune the dissemination tree.

For a given constraint combination $\{\phi_1, \ldots, \phi_1\}$, the false positive rate for the attribute combination $\{\alpha_1, \ldots, \alpha_k\}$ during a time window t_w can be counted directly as:

$$\mathrm{fp}\left(\{\alpha_1, \ldots, \alpha_k\}\right) = \frac{\text{number of false positives for } \{\alpha_1, \ldots, \alpha_k\} \text{ during } t_w}{t_w} \tag{5.2}$$

In other words, whenever a router identifies a message as a false positive, it increases a counter for this particular attribute combination. In theory, this would require counting false positives for the power set of the possible context attributes. In practice, however, we expect the number of relevant attribute combinations, i.e., those that are actually used for addressing messages, to be much lower. We further discuss the overhead of these statistics in Section 5.3.2.

For instance, assume a ContextRouter receives 749 messages with constraints on the attributes $\{\alpha_1, \alpha_2, \alpha_5\}$ over a 10 s window. Of these, 146 are false positives, i.e.

the node discards the messages without forwarding them or delivering them to a locally connected client. Thus, the false positive rate for this attribute combination is:

$$\mathrm{fp}\left(\{\alpha_1, \alpha_2, \alpha_5\}\right) = \frac{146}{10\,\mathrm{s}} = 14.6\,\frac{1}{\mathrm{s}}.$$

Context Update Rate

The *context update rate* u reflects the amount of load that results from propagating and updating a certain piece of context information.

Similar to the false positive rate, it can be observed directly for an existing composite context C as:

$$\mathrm{u}\left(C\right) = \frac{\text{number of updates for } C \text{ during } t_w}{t_w} \tag{5.3}$$

However, when considering an attribute set for propagation from the attribute combinations that are used in addressing, it is necessary to estimate the resulting update rate: since no composite context with this attribute combination exists, yet, it cannot be observed directly. In this case, the update rate can be estimated from the update rates of each of the attributes in the set. (From every context update that a router receives, it can derive the changed attribute(s) and increase an update counter(s) for the particular attribute(s).) Formally, the update rate of an attribute set $\mathcal{A}_p = \{\alpha_1, \ldots, \alpha_k\}$ can be estimated as the sum of the updates observed for its individual attributes:

$$\mathrm{u}\left(\mathcal{A}_p\right) = \mathrm{u}\left(\{\alpha_1, \ldots, \alpha_k\}\right) = \frac{\sum_{i=1}^{k} \text{observed number of updates for } \alpha_i \text{ during } t_w}{t_w} \tag{5.4}$$

Furthermore, let us assume that the ContextRouter from the example in the previous section observes 17 updates for α_1, 36 for α_2, and 45 for α_5. Based on these numbers, it can estimate the context update rate for this particular attribute combination as:

$$\mathrm{u}\left(\{\alpha_1, \alpha_2, \alpha_5\}\right) = \frac{17 + 36 + 45}{10\,\mathrm{s}} = \frac{98}{10\,\mathrm{s}} = 9.8\,\frac{1}{\mathrm{s}}.$$

As one can see from this example, propagating a composite context for this particular attribute combination would prevent more false positives than it would take to keep the information up-to-date. A ContextRouter that receives such a newly created composite context can then directly observe the update rate for this particular context.

Distribution Prune Rate

The *distribution prune rate* p is a measure of the usefulness of a piece of context information *after* it has been propagated.

When a node propagates context information to a neighbor, it no longer receives the corresponding false positive messages; after all, this is the reason behind the propagation of the information. Context constraints in messages may change over time, though. This can lead to the messages that originally caused the propagation of some context information to no longer occur in the network. Unfortunately, it is impossible for a node to determine whether a lack of false positives is the result of the information it propagated or the result of a change in message addressing. Thus, a node may still update context information it propagated to a neighbor even though the false positive messages it is supposed to stop no longer occur regularly (or even at all).

To determine whether a context C is still useful, we therefore measure its distribution prune rate at the pruning node by counting the number of messages that are pruned using the information of C:

$$\mathrm{p}(C) = \frac{\text{number of pruned messages for } C \text{ during } t_w}{t_w} \tag{5.5}$$

Coming back to the example from the previous sections, we further assume that another ContextRouter has received a composite context C_{comp} for $\{\alpha_1, \alpha_2, \alpha_5\}$ from a neighbor. In a later 10 s time window, this node observes a distribution prune rate of 37 messages for this context, resulting in a prune rate of:

$$\mathrm{p}(C_{\text{comp}}) = \frac{37}{10\,\mathrm{s}} = 3.7\,\frac{1}{\mathrm{s}}.$$

If the observed update rate remained largely unchanged at, e.g., $9.3\,\frac{1}{\mathrm{s}}$, we can see that the context is no longer useful, as keeping it updated now far outweighs any savings from not forwarded false positives.

Based on these considerations and measurements, we are introducing a per-link adaptive context propagation approach in Section 5.3.3.

Smoothing

To limit the influence of short term changes in messages and updates, we smooth the measurements over time. Since a node may maintain a rather large number of candidates, we aim to limit the amount of history needed for these measurements. We therefore use an exponential moving average with a factor β to smooth our measurements, since its calculation only requires the current measurements as

well as the previously calculated value. Thus, e.g., the smoothed false positive rate $\mathrm{fp}_{S,t}(\mathcal{A})$ for an attribute set $\mathcal{A} = \{\alpha_1, \ldots, \alpha_k\}$ at time t is calculated from the observed values at time t and previously smoothed values from time $t-1$ as:

$$\mathrm{fp}_{S,t}(\{\alpha_1, \ldots, \alpha_k\}) = \beta\, \mathrm{fp}_t(\{\alpha_1, \ldots, \alpha_k\}) + (1-\beta)\, \mathrm{fp}_{S,t-1}(\{\alpha_1, \ldots, \alpha_k\}) \tag{5.6}$$

The same applies to the calculation of the smoothed context update rate $\mathrm{u}_{S,t}(C)$ and the smoothed distribution prune rate $\mathrm{p}_{S,t}(C)$.

By adjusting β, an administrator can place more emphasis on the measurements of the current time windows, thus reacting quickly to changes, or they can reduce the influence of the current window, thus smoothing out the measurements with past observations.

Overhead of Measurements

The impact of the statistic on the routers is minor. For the false positive rate, an implementation can use, e.g., Bloom filters to efficiently lookup and store false positive numbers for the observed attribute sets. Also, the nodes can regularly remove information about attribute sets, for which they have not registered a false positive in some time. This cleans out rarely addressed attribute combinations and keeps size of the statistic manageable.

The statistics for contexts that a node has received increases the size of each context by a fixed amount: two counters for the number of prunes and updates in the current observation window and two floats for the respective rates that are updated after the window ends.

Additionally, each router maintains an update rate for each individual attribute that is used in contexts. This is used to estimate an update rate for an arbitrary set of attributes. Since there is a fixed number of defined attributes in a Contextcast system, the space requirements for the individual attribute update rate statistic is also limited.

5.3.3 Per-link Adaptive Context Propagation

Using the measurements we describe in the previous section, we can now present the details of our adaptive context propagation algorithm, the necessary changes in message forwarding and our approach to invalidate old context information.

Benefit of Context Information

As we described in Section 5.3.2, each node observes for each composite context how often a message was not forwarded due to the information in the composite

context, i.e., the prune rate. Additionally, it monitors the update rates for all the composite contexts it has received.

Using a composite context's prune rate as well as its update rate, each node can calculate the benefit of a certain piece of context information that it has received from a neighboring router.

Definition 5.3 (Benefit of a Composite Context)**.** Let C_{comp} be a composite context. Using Equation 5.5 and Equation 5.3, a router can calculate the *context benefit* $\mathrm{B}(C_{\text{comp}})$ as the smoothed distribution prune rate $\mathrm{p_S}(C_{\text{comp}})$ over the smoothed context update rate $\mathrm{u_S}(C_{\text{comp}})$:

$$\mathrm{B}\left(C_{\text{comp}}\right) = \frac{\mathrm{p_S}(C_{\text{comp}})}{\mathrm{u_S}(C_{\text{comp}})} \tag{5.7}$$

Intuitively, the context benefit gives us a measure for the amount of messages saved by a composite context C_{comp} versus the load that is necessary to keep the information up-to-date.

This works well for a composite context that a node has received from a neighbor. However, when considering a set of attributes $\mathcal{A}_{\text{cand}} = \{\alpha_1, \ldots, \alpha_k\}$ for propagation, a node can neither determine the prune rate nor the update rate directly. Thus, we need to approximate both numbers using the rate of false positives it receives and the estimated update rate for the attribute set, derived from the observed update rates of individual attributes. Using these two numbers, we can estimate the benefit of propagating a composite context for a given attribute set.

Definition 5.4 (Benefit of a Composite Context for a Candidate Attribute Set)**.** Let $\mathcal{A}_{\text{cand}} = \{\alpha_1, \ldots, \alpha_k\}$ be an attribute set that is considered for propagation. Using Equation 5.2 and Equation 5.4, a router can calculate the *attribute set benefit* $\mathrm{B}(\mathcal{A}_{\text{cand}})$ as the smoothed false positive rate $\mathrm{fp_S}(\mathcal{A}_{\text{cand}})$ over the smoothed update rate $\mathrm{u_S}(\mathcal{A}_{\text{cand}})$:

$$\mathrm{B}\left(\mathcal{A}_{\text{cand}}\right) = \frac{\mathrm{fp_S}(\mathcal{A}_{\text{cand}})}{\mathrm{u_S}(\mathcal{A}_{\text{cand}})} = \frac{\mathrm{fp_S}(\{\alpha_1, \ldots, \alpha_k\})}{\mathrm{u_S}(\{\alpha_1, \ldots, \alpha_k\})} \tag{5.8}$$

Obviously, only composite contexts with $\mathrm{B}(C) > 1$ benefit the load of the system, as they enable routers to prune more messages than the load generated by their respective updates. The same applies to candidate attribute sets $\mathcal{A}_{\text{cand}}$: the composite contexts for those with $\mathrm{B}(\mathcal{A}_{\text{cand}}) > 1$ save more false positives than the load that is required to keep the resulting composite context up-to-date. We show in the following section how ContextRouters use these numbers to decide which context information to propagate as a composite context.

Selecting Attribute Sets for Propagation

The previous section described a measure for the benefit of propagating some context information. The nodes still need a way to actually select an attribute set for the propagation of a composite context. Obviously, it is not scalable to compute the power set of all attributes and consider each element, i.e., every possible attribute combination. Thus, the routers need a more efficient method to select candidate sets for propagation.

For this reason, our selection of candidate attribute sets is based on the observed sets of attributes used in constraints and their respective propagation benefit: After each time window, every router updates its local statistics for the received false positives. Then, it calculates the expected propagation benefit $B(\mathcal{A})$ for each attribute set that is recorded in its statistics using Equation 5.8.

For those attribute sets that have a benefit above a *propagation threshold* $B_{th,P}$ it then creates a composite context and propagates that towards the neighbor. The propagation threshold limits context propagation to those candidates that actually offer an (adjustable) benefit over simple speculative forwarding. An administrator should observe the fluctuation in the calculated benefits in a given system and derive a suitable propagation threshold $B_{th,P} > 1$. In particular, it should be high enough that these fluctuations do not result in $B > B_{th,P}$ in one cycle and $B < B_{th,P}$ (or worse, $B < 1$) closely afterwards. Otherwise, the system propagates candidates where, in one time window, the false positives outweigh the updates enough to justify propagating a composite context; and in the next time window, the updates dominate, thus contradicting the decision to propagate the candidate set.

Additionally, a newly propagated attribute set $\mathcal{A}_{new}$ may be a superset of another attribute sets $\mathcal{A}_k$. Since a composite context for $\mathcal{A}_{new}$ contains all the information of composite contexts for its subsets, a node can then stop updating the composite context for $\mathcal{A}_k$. The node receiving the new composite context can also determine all subset composite contexts and remove them from its routing table. These steps are summarized in Algorithm 5.3

After such a propagation, the node must update the composite context in the future; at least until it is invalidated because the messages have changed sufficiently so the information is no longer useful.

Message Forwarding

As we hinted at before, this requires only very minor changes to the message forwarding algorithm. Algorithm 5.4 shows a modification of Algorithm 3.2, which can handle incomplete context knowledge and composite contexts: Routers perform a

Algorithm 5.3 Composite Context Propagation

Require: A list FPstats of candidate attribute sets for neighbor N.
Ensure: Composite context for the attribute sets with sufficient benefit propagated to N.

```
C_prop ← ∅                    ▷ For which attribute sets to propagate a composite context
for all A_cand,i ∈ FPstats do
    if B(A_cand,i) > B_th,P then
        skip ← false
        for all A ∈ C_prop do
            if A_cand,i ⊆ A then ▷ Already going to propagate a superset of A_cand,i
                skip ← true
                continue
            else if A ⊆ A_cand,i then
                C_prop ← C_prop \ A
            end if
        end for
        if not skip then
            C_prop ← C_prop ∪ A_cand,i
        end if
    end if
end for
for all A ∈ C_prop do
    Propagate a composite context for A to N
end for
```

directed forwarding if they have the necessary knowledge to evaluate the constraints of a Message M. Otherwise, they fall back to speculative forwarding.

Algorithm 5.4 Message Forwarding with Composite Contexts

Require: A message M, received from neighbor N_M or a local client.
Ensure: M forwarded to at least the same neighbors as Algorithm 3.2.

```
boolean array[Neighbors] directed ← [false,...,false]
boolean array[Neighbors] forwarded ← [false,...,false]
for all (C_comp, N_C) ∈ Routingtable do                    ▷ Directed forwarding
    if forwarded[N_C] = true ∨ N_C = N_M then
        continue
    end if
    if Attributes(M) ⊆ Attributes(C_comp) then             ▷ Sufficient knowledge
        directed[N_C] ← true                               ▷ for a directed forwarding to N_C
        if C_comp ⊏ M then
            Forward a copy of M to N_C
            forwarded[N_C] ← true
        end if
    end if
end for
for all N ∈ Neighbors do                                   ▷ Speculative forwarding
    if directed[N] = false then                            ▷ for remaining neighbors
        Forward a copy of M to N
    end if
end for
```

If a node knows several composite contexts $C_{comp,1}, \ldots, C_{comp,k}$, such that $\forall i \in \{1, \ldots, k\} : \texttt{Attributes}(M) \subseteq \texttt{Attributes}(C_{comp,i})$, it can use any of these contexts to evaluate M's constraints. They all contain complete knowledge for their respective attribute set, i.e., in particular also the attributes that are constrained in M. Algorithm 5.4 uses the first one in the routing table.

Obviously, the evaluation whether $C_{comp} \sqsubset M$ depends on the structure of C_{comp}. In the case of a simple list of partial contexts, as we discussed in Section 5.3.1, $C_{comp} \sqsubset M$ if and only if

$$\exists C_{partial} \in C_{comp} : C_{partial} \sqsubset M.$$

Context Invalidation

After some time, the false positive messages that a composite context prevented may no longer occur (or only very rarely). In this case, the continuing updates of

this composite context by its originating node cause unnecessary load in the system. We must therefore invalidate such a composite context, which again lowers the overall system load. Obviously, the messages this context stopped, should they occur occasionally in the future, are then propagated speculatively, potentially leading to an increased false positive load.

Similar to the propagation of a composite context, a router sends back an invalidate message for a composite context whose benefit drops below a configurable *invalidation threshold* $B_{th,I}$. E.g., a value $B_{th,I} < 1$ causes the invalidation of all those contexts that cause more update load than the false positives they prevent. This is shown in detail in Algorithm 5.5.

Algorithm 5.5 Composite Context Invalidation

Require: The list of all composite contexts received from neighbors.
Ensure: All contexts invalidated for which the updates dominate the system load.

```
for all (C_comp, N_C) ∈ Routingtable do
    if B(C_comp) < B_th,I then
        Send invalidation message for C_comp to N_C
    end if
end for
```

Propagation/Invalidation Hysteresis

If the two values for the propagation threshold $B_{th,P}$ and the invalidation threshold $B_{th,I}$ are too close or the same, the system would become unstable and respond to small changes in message or update behavior. These small fluctuations might raise or lower a given context's benefit above or below the threshold, causing a propagation or invalidation, respectively. Having these two values configurable separately serves as a hysteresis for the propagation and invalidation of composite contexts.

This way, an administrator can configure the system to only propagate a composite context if it saves, e.g., 30% more false positives than the updates it causes ($B_{th,P} = 1.3$). And the same administrator might also configure it to invalidate those contexts which save 10% less false positives compared to the necessary updates ($B_{th,I} = 0.9$).

5.4 Evaluation

5.4.1 Setup

We have implemented our adaptive approach, shown in Algorithm 5.3, Algorithm 5.4, and Algorithm 5.5, in a prototype implementation to evaluate its performance. As the goal of the presented approach is a reduction of the overall load on the ContextRouters in the overlay network, we have implemented our prototype in the PeerSim [MJ09] network simulator.

For the simulation, the network consists of 500 ContextRouters, uniformly placed on an area of $[0,1] \times [0,1]$ and connected according to the Heuristically Optimized Trade-off model (cf. [FKP02]). The nodes are added sequentially, the first being the root node of the network, n_0. A newly added node n_i connects to a previously added node n_j, which minimizes the weighted distance metric $\gamma \cdot d_{ij} + h_j$, where d_{ij} is the Euclidean distance between the nodes n_i and n_j and h_j is the network distance, i.e., the number of hops, from node n_j to the root node n_0. We selected the weight parameter $\gamma = \sqrt{n}$ for a network of n nodes (the authors of [FKP02] use the letter α, which designates an attribute in Contextcast). This leads to an acyclic undirected network graph, in which the node degrees exhibit an Internet-like power-law distribution (cf. [FKP02]).

A fraction of 60 % of these nodes are then selected as access nodes (ContextNodes). They get assigned a rectangular service area with edge lengths between $[0.05, 0.06)$. Each service area is then placed on the simulation area with its corresponding ContextNode in the center. Areas extending beyond the $[0,1]$ interval in either direction are cut off to remain within the boundaries of the simulation. This leaves a number of routers with degree 1 without an access network. Even though no client can ever connect to such a node, they can still send messages, e.g., a provider of commercial messages might operate its own overlay node as gateway. Thus, they are still useful and not a dead end in the network.

We simulate an average of 9 000 clients in the system, each with a location identical to its access network's service area (cf. Chapter 4). Additionally, each contains between six and twelve numeric attributes, uniformly chosen out of a set of 30 different ones. The values of the numeric attributes follow normal distributions with different means and standard deviations for each attribute.

The network load is determined by the rates of updates and messages. To not favor either update or message load, we simulate update and message rates of 5 per simulation cycle, each.

Updates are generated according to the following instructions, which models clients both changing their context and clients leaving and new ones joining the system: A fraction of 70 % of all updates results from a change of a single, uniformly

Parameter	Value
# of routers	500
γ	$\sqrt{500}$
# of access nodes	300
# of clients	9 000
# of numeric attributes/client	6–12 (uniform distribution)
average update rate	5/cycle
# of numeric constraints/message	1–3 (uniform distribution)
average message rate	5/cycle

Table 5.1: Summary of simulation parameters

selected attribute. If the changed attribute is the client's location, the simulation uniformly selects one of its ten nearest neighboring ContextNodes as its new access node; the value of a numeric attribute is changed in the same manner as it is created for new contexts. For the remaining 30 % of updates, a uniformly selected context disconnects and a different random context—created in the previously described way—connects to the system.

Context messages originate from $\frac{1}{10}$ of our routers, which are chosen as senders uniformly from all routers. They are created in a manner similar to context updates: A fraction $\lambda = 0.5$ contains a target location with edge length between 0.2 and 0.3. In addition, they contain between one and three numeric constraints; the selection of the numeric constraints follows a Zipf distribution, with parameter $s = 1.5$ and using different attribute permutations for the different senders. This ensures that messages of a sender show a bias towards similar addressing, which is the underlying assumption behind our adaptive approach.

For our evaluation, we compare our adaptive algorithm (designated *A* in the results) with a baseline approach (designated *B*), which corresponds to the reference algorithm from Section 3.3.3 and propagates all context updates into the network. We look at both the steady-state and dynamic performance of our adaptive approach. For each experiment, we run ten simulations with different random seeds and then compute the arithmetic mean of the simulation runs. Unless stated otherwise, the standard deviation of the different runs is negligible. Table 5.1 contains an overview of our setup.

5.4.2 Load Reduction: Impact Of Propagation Threshold

To illustrate the merit of our approach, we compare the adaptive context propagation with the baseline over a sampling period of 1 000 simulation cycles. We observe the

system in a steady state to ensure that the initial propagation of client contexts does not put the baseline approach (see Section 3.3.3) at a disadvantage.

Figure 5.3 shows the update and message load for the adaptive and the baseline approach for various *propagation thresholds* $B_{th,P}$. The *invalidation threshold* $B_{th,I}$ was set to a constant 0.9 for this experiment.

The results show that our adaptive approach reduces the overall load by between 42 % and 43 % compared to the baseline algorithm. Looking at the individual load elements, one can see that the adaptive approach lowers the amount of updates by between 87 % and 94 %. At the same time, since nodes no longer have complete context information, they need to fall back to speculative forwarding for more messages, thus increasing the number of forwarded messages by 614 % up to 709 %. The difference in message load is a result of false positives, of course, since the baseline approach forwards and delivers a message if and only if there is a matching recipient. However, even with this increase in messages, the overall load on the system is almost cut in half, due to the reduction in updates. This meets our expectations, since one of the major goals in the design of this algorithm is to trade off update load against additional false positives, while preserving the "no false negatives" semantics. Context invalidations play virtually no role in this experiment since the system is in a steady state and messages do not change significantly enough.

The propagation threshold $B_{th,P}$ determines what context information is forwarded between routers. The higher this number, the fewer information neighbors exchange, thus increasing false positive load, while at the same time reducing the amount of update load. In this scenario, with each node using a relatively similar attribute set for all its messages, the system soon reaches a steady state: all nodes have propagated composite contexts for the prevalent messages in the system. The attribute sets of these composite contexts have a high benefit value. Thus, with a lower threshold, the system establishes additional composite contexts for rarely addressed attribute sets. They increase update load without actually pruning many additional distribution trees; higher thresholds removes these from the system without causing a large number of additional messages, thus lowering the overall system load. For very high thresholds, the effect reverses and the additional false positives start dominating the total system load.

5.4.3 Load Reduction: Impact Of Message & Update Rates

As the previous experiment showed, our adaptive approach reduces update load at the expense of an increase in message load. If a given percentage of all forwarded messages are overhead in the form of false positives, increasing the message sending rate also increases the absolute amount of false positives. To investigate the influence

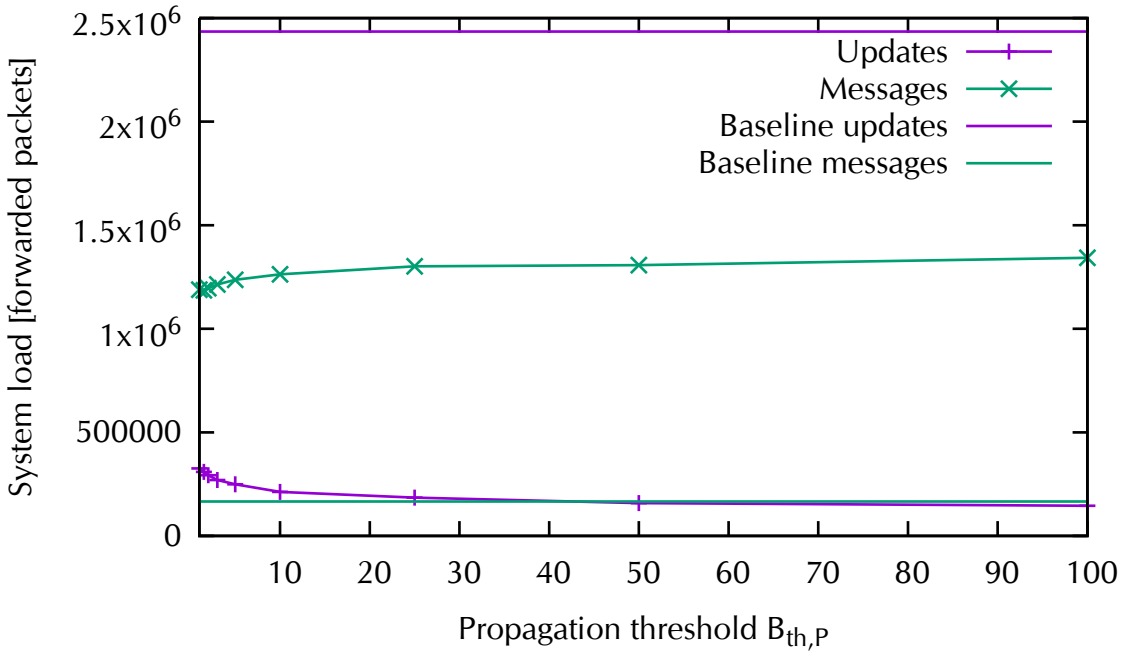


Figure 5.3: System load for various values of the propagation threshold $B_{\text{th,P}}$

of the message rate on our system, we observe the system with $B_{\text{th,P}} = 1.5$, a constant update rate $r_u = 5/\text{cycle}$ and message rates $r_m \in \{5, 7.5, 10, 12.5, 15\}$, for 1 000 cycles in a steady state.

Figure 5.4 shows the amount of messages and updates for the different message rates, for the adaptive (A) and baseline approaches (B). As the figure shows and as one would expect, higher message rates increase the message load on the system, both legitimate and false positive messages. Once the message rate reaches 2.5 times the update rate, the overall load of the adaptive approach is higher than the overall load of the baseline algorithm. This is hardly surprising, since our algorithm is tailored more to handling dynamic context attributes such as location or mode of transportation. It is less well suited for rather static attributes (e.g., subscriptions to news feeds).

Also, the figure shows that the higher message load causes an increased number of updates for the adaptive approach. While this may seem strange, considering the constant update rate in the experiment, the effect can easily be explained. Due to the higher message rates, more attribute combinations reach the propagation threshold and thus the routers create composite contexts for these combinations.

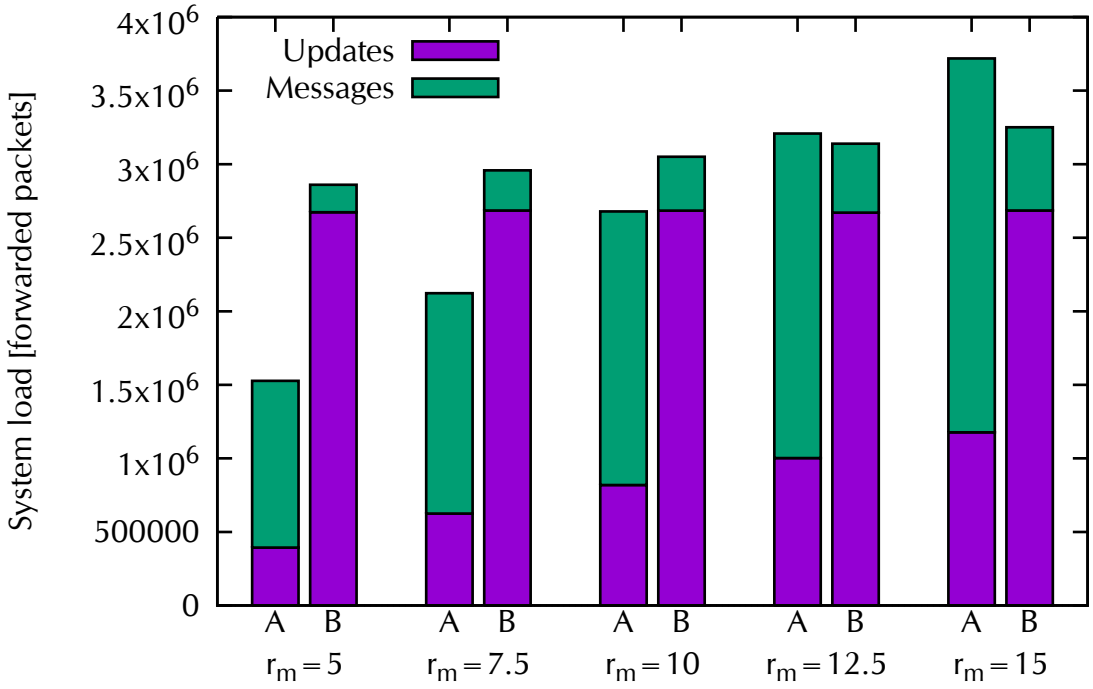


Figure 5.4: System load for update rate $r_u = 5$ and various message rates r_m

5.4.4 Stabilization: Impact Of Exponential Moving Average

The continuous measurements enable our system to adapt to changes in the addressing of messages. To measure how quickly our system adapts, we introduce a drastic change after 2 500 simulation cycles: As we explained in Section 5.4.1, every sender constructs messages using a certain preference of attributes. Technically, this is implemented as a Zipf distribution over a permutation of attributes. In this experiment, for the first 2 500 cycles, the nodes are limited to select numeric constraints from the first 15 attributes. At 2 500 cycles, each node selects a new permutation of the attributes from the second half of the available 30 attributes. Thus, at this instant, the complete addressing in the system changes, requiring the distribution of new composite contexts to match the new messages.

To measure the impact of the parameter β of the exponential moving average, we vary this parameter between 0.5 and 0.8, with a constant window length of 100 cycles. Figure 5.5 shows the amount of messages and new composite contexts over time for $\beta = 0.5$ and $\beta = 0.8$. (To maintain the legibility of the figure, we omit the measurements for values other than 0.5 and 0.8; the curves show the same trend, they just differ in the absolute number of messages. Also, the figure shows the average load per cycle for periods of 25 cycles, again to increase legibility by filtering

out short-term changes.)

As we can see for both values, at 2 500 cycles the amount of messages increases sharply due to speculatively forwarded messages; at this time, previously established composite contexts can no longer be used for directed forwarding. It then decreases again as new composite contexts are propagated and used to reduce the speculatively forwarded messages. The higher β shows a quicker reaction, with more new composite contexts and a faster reduction of message load, since the influence of the newest measurement on the moving average is higher. It also exhibits a lower overall number of messages (i.e., speculative forwarding) once the system stabilizes since nodes propagate composite context faster; with a lower value β, the same false positive needs to be observed for several observation windows before the benefit is high enough to actually propagate a composite context. Note that, due to the window length of the adaptive approach of 100 cycles, updating the statistics and establishing new composite context happens only every 100 cycles. Therefore, the graph shows composite context load as spikes every 100 cycles and no composite contexts in between that.

An administrator should set β to the highest possible value that still sufficiently filters out transient changes in message addressing. This choice allows for both a quick adaptation to changes and an overall lower load due to reduced speculative forwarding.

5.4.5 Stabilization: Impact Of Window Length

We also observe the stabilization for different window lengths $t_w \in \{25, 50, 75, 100\}$ (in simulation cycles). We introduce the same change to the system as in the previous section, i.e., after 2 500 simulation cycles all senders radically alter their message patterns. Thus, the previously established composite contexts become useless and the routers need to propagate new ones.

Figure 5.6 shows how the window length t_w influences the time it takes for the system to react to changes. Again, for legibility, we limited the figure to $t_w = 25$ and $t_w = 100$. (Note that, since we are showing average load for 25 cycle periods as before, for $t_w = 25$ we see new composite contexts established in every 25 cycle period. In contrast, we only see a spike in composite contexts for the first 25 cycle period for $t_w = 100$, as statistics are only updated every 100 cycles.) A higher window length, i.e., 100 cycles, requires a longer period of speculatively forwarded messages after the change. This is due to the fact that with $t_w = 25$, the nodes can establish the first new composite contexts more quickly, after only 25 cycles, which then immediately reduce message load. With $t_w = 100$, in contrast, the nodes need to speculatively forward for 75 more cycles before establishing new composite contexts. However, for $t_w = 100$, we also see that the system establishes more new

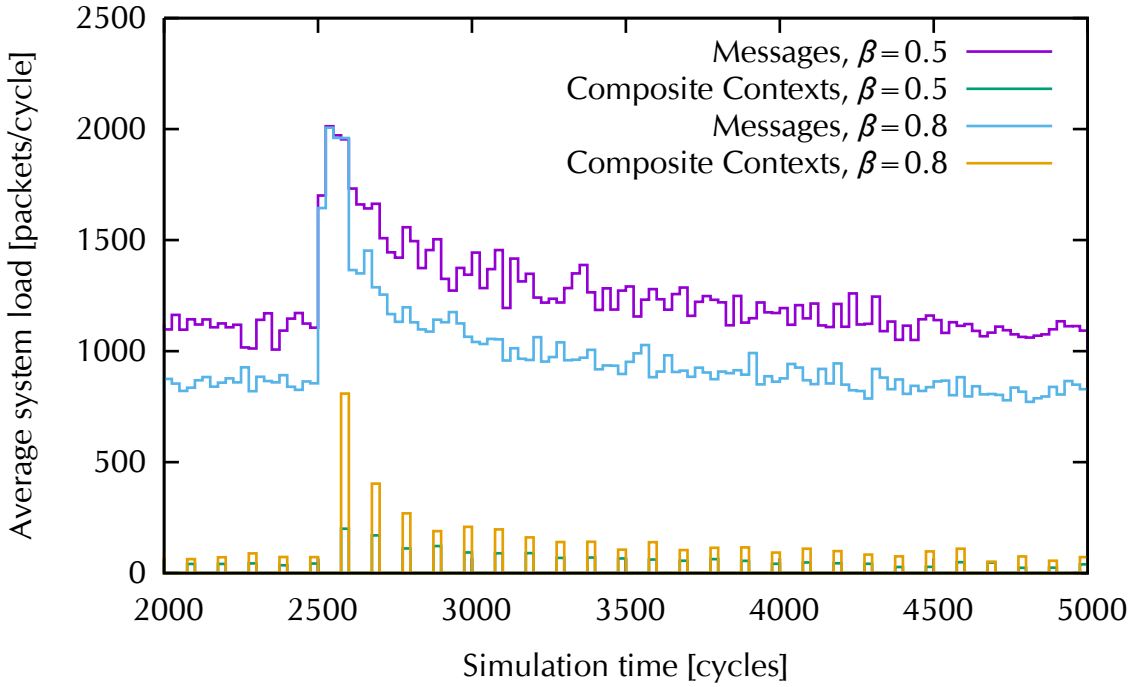


Figure 5.5: Stabilization for different values of β

composite contexts over a 100 cycle period. Thus, even though a drastic change causes more speculatively forwarded messages for higher t_w at first, the number decreases faster as well and reaches an overall lower amount. This is caused by a better observation of messages over the longer time window.

Just as for β, we recommend that an administrator sets t_w to the highest value that offers a good balance between agility when reacting to changes and longer observations and thus more accurate statistics. Obviously, this depends on the rate of messages and updates that nodes observe. E.g., if a router receives only 5 messages per second, the window needs to be longer than if it receives 1 000 messages per second.

5.4.6 Analysis

Recall, the goal of the adaptive propagation of contexts is a reduction of context broadcast load in the network, at the cost of message broadcast load.

Our approach adapts to the actual messages and contexts in a system. Let us assume that on average, a fraction α of all messages is sent speculatively. These messages find a matching recipient and are not false positives or the rate of false

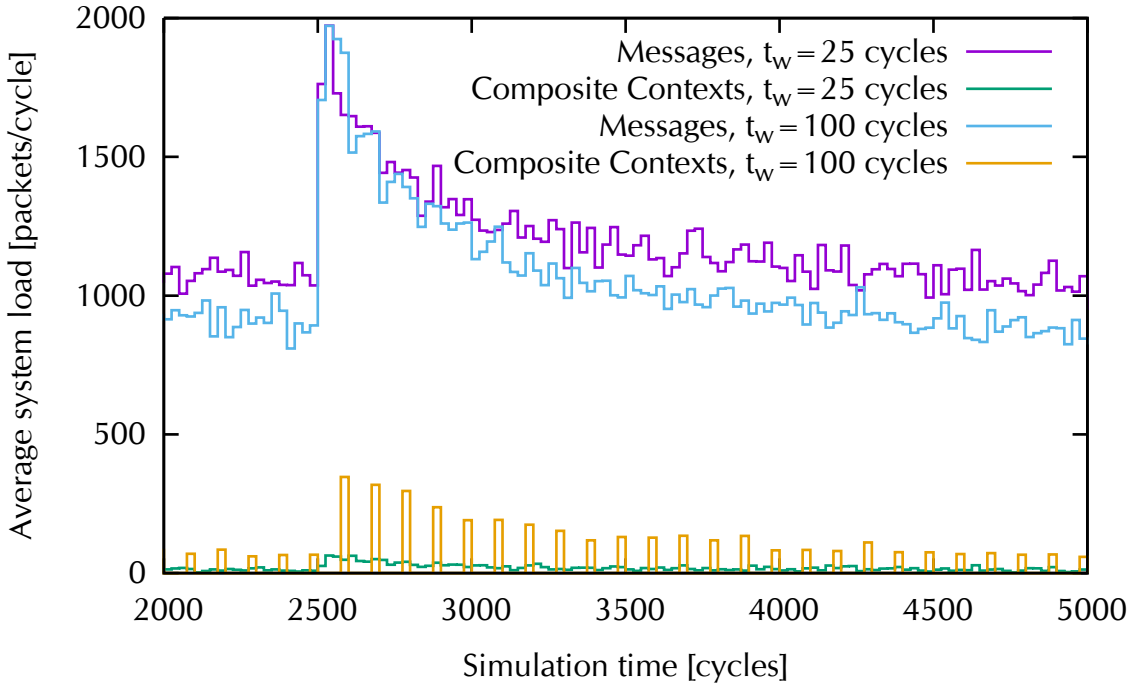


Figure 5.6: Stabilization for different window lengths t_w

positives is below the context update rate. This, in turn, means that a fraction $(1-\alpha)$ of messages is sent directed and thus the result of propagated composite contexts.

For the broadcast messages, the system load caused by context updates is 0. At the same time, the broadcast messages incur at most a false positive load that is below the context update rate $u\frac{1}{s}$. Otherwise, the algorithm would simply propagate a composite context and thus move that particular message type into the other group. For the directed messages, the system incurs the previously mentioned context update rate of $nmu\frac{1}{s}$.

Combining these two results leads to an overall system load from false positives and composite context updates of

$$\text{Average forwarding load} = (\alpha nu + (1-\alpha)nmu)\,\frac{1}{s}.$$

5.5 Related Work

As we discussed in Section 2.3, the idea of context-based routing (or Contextcast) is similar to content-based routing. For scalability reasons, content-based Pub/sub systems such as Siena [Car98, CRW00], REBECA [Müh01], PADRES [FJLM05], etc.,

are built as distributed systems: a network of brokers delivers notifications from sources to all interested, i.e., subscribed, clients.

The author of [Müh02] distinguishes various routing strategies for content-based networks; most other systems have adopted these or similar schemes. They all require that certain information is broadcast in the network, to allow event delivery between arbitrary nodes. The methods differ in what type of information is propagated in the network, though.

- The simplest method floods events in the network, thus every event reaches every broker and also every client connected to a broker. The clients can then filter out events that they are not interested in. Obviously, broadcasting events scales badly in systems with high event rates.

- The second scheme, routing based on subscriptions, floods the information about client subscriptions to the brokers. Thus, every broker can use the subscriptions to determine to which neighboring broker(s) it needs to forward an event. This is conceptually identical to the directed forwarding in Contextcast, which we describe in Section 3.3.3. It is based on the assumption that subscription changes occur less frequently than events. In this case, the scalability is increased by flooding the lower rate subscriptions and then disseminating events only towards brokers with a matching subscription from one of their clients.

  Optimizations allow to reduce redundancies in client subscriptions, such as removal of identical subscriptions, cover, or merge. But the basic principle of propagating the information of all subscriptions to all brokers remains the same. We have shown a similar approach for Contextcast with the propagation of coarse context information in Chapter 4.

- Third, the flooding of subscriptions can still cause a rather high load in the network. The use of *advertisements* provides a further improvement to this scheme [Car98]. Advertisements describe the notifications that a source sends out. Each source of events broadcasts advertisements for all its events. With advertisements, brokers can forward subscriptions only towards those sources that may produce a matching event. Thus routing paths are established only between sources and clients with subscriptions to their respective events. Again, since the system now broadcasts advertisements, the assumption is that advertisements do not change as fast as subscriptions.

In Pub/sub systems, advertisements proved to significantly improve the scalability [MFGB02]. Since Contextcast routing is similar to the filter-based routing

strategies of content-based networks, it stands to reason that Contextcast also benefits from a description of possible contextual messages. With such an optimization, ContextRouters can propagate context information only towards sources of messages that a given context matches.

Contextcast, however, offers support for senders that send only a single message of a particular type. In such a scenario, advertisements as used by Pub/sub offer no tangible benefit, since broadcasting the message places essentially the same load on the network as broadcasting an advertisement. And after an advertisement, the routers must still propagate contexts and finally disseminate the message, adding further to the load in the network.

The adaptive propagation of client contexts, which we introduced in this chapter, is a generalization of advertisements in Pub/sub. We use the observation of speculatively forwarded false positives as a form of implicit advertisements: Without context information, ContextRouters must forward speculatively to preserve Contextcast's delivery semantics. These speculatively forwarded messages represent the messages that occur in the system. Thus, instead of explicit descriptions of messages, ContextRouters rely on the observed false positive messages. Using these, contexts can be propagated towards the sources of messages. The nodes that possess the necessary context information to evaluate the constraints of a message can use a directed forwarding.

Additionally, advertisements in Pub/sub systems optimize the establishing of subscription paths statically: The propagation of subscriptions is limited towards sources of matching events, but all subscriptions get propagated. The process does not take the actual load generated by subscriptions and events into accounts. For instance, for events of a particular advertisement that occur only rarely, it might be better to broadcast these events than to forward quickly changing subscriptions towards the source. Our adaptive approach, in contrast, bases its context propagation decision on both the rate of messages as well as the rate of context updates. This way, a propagation of client contexts in Contextcast takes place only when it benefits the overall system load.

5.6 Summary

In this chapter, we presented an approach to increase the scalability of context-aware routing by adaptively propagating context information. The adaptive propagation of routing information reduces the network load caused by context updates. It is a replacement for the reference algorithm we introduced in Chapter 3. To maintain the semantics of Contextcast, it ensures that a message reaches at least the same recipients as when disseminated using the reference algorithm.

The algorithm achieves this by propagating context information that can be used for a directed forwarding for types of messages that occur often. This way, routers have the necessary context information to prune the distribution tree for these frequent message types. Message types that occur rarely, in contrast, are forwarded speculatively, since it would place more load on the system to keep the required context information up-to-date than simply suffer an occasional false positive for such rare messages.

To this end, the routers record statistics for incoming messages and updates. From these numbers, each router can deduce how many false positives a piece of context information prevents and how many updates are required to keep it updated. The quotient of these numbers is the benefit of a piece of context information. The adaptive propagation algorithm uses this benefit to propagate composite contexts to neighbors if their benefit exceeds a configurable *propagation threshold*. At the same time, if a certain class of messages no longer occurs, the benefit of the corresponding context information is low; it requires more updates than the false positives it supposedly prevents. In this case, a router can invalidate a piece of context information once its benefit falls below an also configurable *invalidation threshold*.

Our evaluation of these concepts have shown that they reduce the overall system load by over 40 % compared to the reference dissemination algorithm. This reduction is largely a result of the drastic reduction of context updates for client contexts that are rarely addressed. This reduction is offset to a certain degree by false positives that occur due to the necessary speculative forwarding.

We have also shown that the system can react if the prevalent message types change over time. An administrator can adjust the parameters of the algorithm to react more quickly to such changes or to better filter out single bursts of a particular type of messages, depending on the observed messages and their changes over time.

Chapter 6

Temporal Addressing in Contextcast

> Time is at once the most valuable and the most perishable of all our possessions.
>
> *(John Randolph)*

6.1 Overview

Contextcast allows to efficiently disseminate messages to clients with a specific context. Due to the complexities of determining the temporal relation between two events in a distributed system, e.g., a client registering a context and another one sending a contextual message, Contextcast so far uses an implicit temporal semantic. Figure 6.1 illustrates this with three contexts C_1, C_2, and C_3, each registered with the system during a certain time interval (marked for C_1 in the example with t_1 and t_2). Only the context C_1 can match a message M sent at time t_M in our previously described Contextcast system; C_2 was deregistered before t_M, thus the client is no longer connected, which prevents message forwarding and delivery. In contrast, the system has no knowledge about C_3 yet and for obvious reasons does not store the message for a later delivery. Even in the case of C_1, M might not reach the client—despite it being sent while C_1 is registered—because the information was not propagated to the necessary routers before t_M. We have discussed the reasons for this in Section 3.3 and introduced the notion of a localized perfect dissemination.

However, explicit temporal specifications open up a number of new use cases: For example, the person writing the minutes of a meeting can distribute the finished, digital version via a Contextcast message to all people who attended the meeting. Such a message would be addressed to "the people that were in room 0.108, yesterday, between 13:00 and 15:00". Another example is a fashion store, which uses Contextcast to send out a questionnaire as part of its marketing strategy. The message is addressed to all "people passing by during the next three weeks starting the next day, are female, and are between 15 and 35 years old". This is the

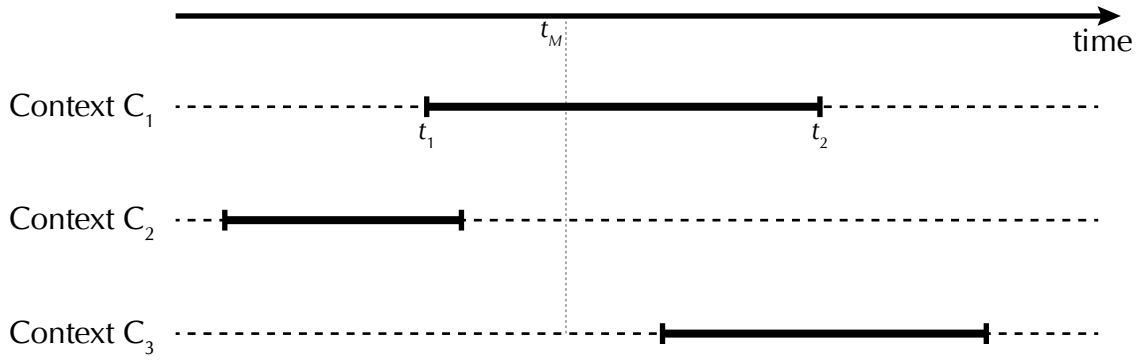


Figure 6.1: Validity periods of three contexts C_1, C_2, and C_3 over time

audience that the store caters to and whose feedback can be used to improve the shopping experience and hopefully also the store's revenue.

As we discussed in detail in Section 2.3, such information distribution is not possible using a Pub/sub system or a Multicast approach: it would require that the participants explicitly subscribe to future notifications or join a corresponding group for many different kinds of messages. While this might be possible for the example of a meeting, it is obviously not practical except for very few subscriptions. And it is definitely not possible for the multitude of other possible messages one might conceive such as the store example. In contrast, Contextcast's sender-centric approach does not require clients to explicitly register for messages they might be interested in; the sender determines the intended recipients of a message with the addressing constraints of a message.

The temporal extension to Contextcast, which we present in this chapter, greatly increases the flexibility of our system. It allows to address contexts that—in addition to other constraints—satisfy a given temporal constraint, both in the past and the future. Such a temporal constraint effectively decouples senders and recipients temporally: they do not need to be connected to the system at the same time, nor does the approach require an established forwarding path of matching context information between the sender and recipient. Note, however, that this is an extension: some messages simply do not require this additional flexibility and may be only relevant at some point in time.

From this initial description, it becomes clear that the previously shown approaches to disseminate Contextcast messages—maintaining context information on routers throughout the system to evaluate context constraints—are not feasible for a temporal Contextcast: To be able to address past contexts, all routers would have to store every context that was ever registered with the system. Additionally, they would need to map these historical contexts to entities in the present. Similarly, for

messages addressing clients with a certain context in the future, the system must store messages until the right time, which the approaches in the previous chapters do not provide.

This leads us to two main challenges that we need to address when extending Contextcast with a temporal aspect: First, the system needs a method to specify the temporal constraints that client contexts must fulfill in order for them to match a message. Second, a temporal Contextcast requires efficient mechanisms to distribute messages to the addressed contexts. Referring back to Figure 6.1, this includes messages that address contexts connected to the system before the time t_M a message M is sent (*historical contexts*, such as C_2) as well as contexts connecting to the system after a message is sent (*future contexts*, such as C_3). We call the messages addressing these two types of contexts *historical* and *future messages*, respectively, and show efficient dissemination approaches for both. Thus, additional measures must prevent such a violation of privacy. When addressing future contexts, messages need to be stored for delivery to a matching recipient some time in the future. Thus, it requires an efficient method to store future messages to ensure, e.g., that messages are not forwarded or duplicated needlessly when no future recipient ever matches a message. Otherwise, future messages may cause a large amount of unnecessary false positive message load.

In the remainder of this chapter, we show the necessary changes to Contextcast for this additional flexibility. We start by describing the requirements for a temporal Contextcast system in detail in the following section. Section 6.3 introduces the changes to Contextcast, a system of temporal constraints that are used for the temporal addressing of Contextcast messages, an archive of historical contexts, as well as the routing algorithms for addressing both *historical* and *future* contexts. We provide an evaluation of the performance and scalability of our approach in Section 6.4. After that, Section 6.5 discusses related work before we summarize this chapter in Section 6.6.

6.2 Requirements

The goal of a temporal extension of Contextcast is to increase the flexibility of the system by allowing messages to explicitly specify the temporal aspect of the dissemination semantic. This complements the existing implicit semantics of disseminating to clients who have currently registered and propagated a context at the time a message is sent. Furthermore, such an extension of the functionality needs to be efficient as to not place too much additional load on the system. This leads us to the following requirements for a temporal Contextcast system:

Temporal Contextcast Semantics. Obviously, if Contextcast is to have the ability to address messages with temporal constraints, it needs a corresponding attribute, with a set of temporal predicates that can be used to constrain matching contexts. These predicates should be as natural as possible to use, i.e., to specify that a time span such as the validity of a context $[t_r; t_d)$ is *before* another one, a predicate "$[t_r; t_d) < [a, b]$" (read as "$[t_r; t_d)$ *before* $[a, b]$)" is easier to use than testing on interval boundaries such as $t_d < a$, particularly for users, but also for application developers.
Efficient Temporal Routing. Depending on the contexts needed to evaluate a temporal Contextcast message, a message falls into one of three categories, which determines its dissemination: *historical messages* address only historical contexts, *future messages* address only contexts in the future, and *hybrid messages* address contexts both in the past and in the future. Both *historical* and *future messages* require efficient routing approaches: First, for historical messages the system needs access to historical contexts, i.e., it requires methods to efficiently store and retrieve historical client contexts (see also the next two requirements) and efficient ways to deliver messages to the corresponding entities even if they are no longer connected. Second, for future messages, it is not known beforehand when or even if a client context will match a message. Thus, the system must take care to not produce false positives in case there is never a matching recipient. This would again waste bandwidth and place unnecessary load on the overlay nodes and links. Finally, hybrid messages can be treated as both a *historical* and *future* message, which can then be processed efficiently by the nodes.
Efficient Retrieval of Historical Context Information. Disseminating messages to historical contexts requires a method to determine matching recipients. The scalability of the approach depends on efficient lookup of historical, matching contexts. We discuss this lookup as part of the routing algorithms for historical messages in Section 6.3.2.
Privacy-aware Storage of Historical Contexts. Archiving historical client contexts is necessary to find the ones matching a given historical message. However, such a history of contexts obviously raises privacy concerns: if an attacker gains access to this data, they may build complete client profiles and abuse the information in various ways. This needs to be addressed in the design of the system; in particular, the system needs to make profiling clients from this stored information difficult.

6.3 Temporal Contextcast

In this section, we discuss the various additions and changes to Contextcast to support temporal constraints in context-aware communication. These are derived from the requirements in the previous section and include (1) a representation of

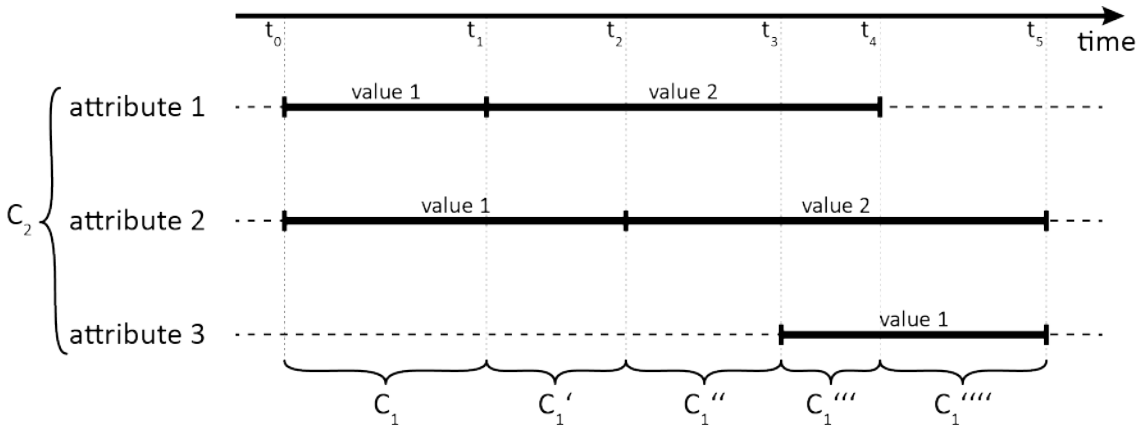


Figure 6.2: Evolution of a sample context C over time

the temporal validity of client contexts as well as a temporal addressing mechanism for Contextcast, which supports both historical and future contexts, (2) an efficient routing strategy for historical messages, together with the necessary privacy-aware archival of client contexts, and (3) an efficient reactive routing approach for future messages.

6.3.1 Temporal Extension for Contextcast

Context Evolution over Time

One of the key aspects of Contextcast and the driving force behind the approaches we have shown so far is the ever-changing nature of user contexts: The attributes that form a user's context are updated constantly and the system needs to account for these changes. Otherwise, Contextcast could simply register a client once, e.g., as part of a multicast group, and then use this information to disseminate all messages.

Figure 6.2 shows an example of the evolution of a single client's context C over time. At time t_0, it is registered with the system, with the attributes 1 and 2 and their respective values. At t_1, attribute 1 changes its value, followed by a new value for attribute 2 at t_2. A third attribute is added to C at t_3. From t_4 on, attribute 1 is no longer a part of C. And finally, at t_5, the context C is deregistered from the system.

From this example, it becomes clear that it is possible to regard the evolution of client contexts in one of two ways:

1. As a sequence of contexts that are valid for a certain period of time; in the example in Figure 6.2, this view is marked as contexts C_1 through C_1''''.

2. As a single context whose attributes are individually marked with timestamps to denote this evolution and the time that each attribute and its value are valid. This view is denoted with C_2 in the Figure.

Functionally, these two alternatives are equivalent: in both cases one can easily determine whether a given context matches a set of constraints including a temporal one. Additionally, a single context with timestamped attributes can be transformed into a sequence of contexts trivially. The other direction is more difficult, though, as the individual contexts need not have any property linking them together, thus making the reconstruction of a single context difficult to impossible.

Privacy-awareness among users and corporations has increased recently, especially after the revelation of global government surveillance programs such as PRISM ([Tox14]). Context-aware systems have access to a large amount of private information such as a user's *location*. It is therefore necessary to take appropriate measures to protect such information, e.g., by partitioning the information and limiting access to the individual parts [Wer15]. Having a single context with time-stamped attributes is equivalent to readily available, long-term context profiles of users. Therefore, we expect that choosing such a temporal context representation would negatively affect Contextcast's adoption. Also, in Contextcast, clients can deregister and newly register a context at any time, without a connection between the two such as a context identifier. Based on these considerations, we treat the evolution of a context as a sequence of individual contexts (C_1 through $C_1{}''''$ in Figure 6.2). To express this fact, we introduce a new context attribute *time*, which models the temporal extent of a context.

Definition 6.1 (Context Attribute *Time*)**.** The attribute *time* of a client context defines the temporal interval $\left[t_{\text{register}}, t_{\text{deregister}}\right)$ the context was registered with the system and thus valid. Its type is "UTC" (Coordinated Universal Time) and a point value can be specified in the format "2014-01-14T09:00:00Z".

For any client context, the interval $\left[t_{\text{register}}, t_{\text{deregister}}\right)$ (or $[t_{\text{r}}, t_{\text{d}})$ for short) is implicitly defined as the time the context was registered with the system (t_{register}) and the time it was deregistered ($t_{\text{deregister}}$). Excluding the deregistration time from the interval prevents ambiguities when contexts change over time: in Figure 6.2, C_1 is valid during $[t_0, t_1)$, $C_1{}'$ during $[t_1, t_2)$, etc. Additionally, since therefore also $t_r \neq t_d$, this definition prevents contexts that have a temporal extent—or duration—of 0, i.e., which would not exist at all.

As before, contexts in Contextcast are registered explicitly but deregistration can happen either explicitly as well, triggered by an application, or implicitly, for instance, because a client loses its network connection. The latter is the case, e.g., when moving into an area without network coverage or when entering a state of

lowered power consumption. Thus, t_r is always known for any given client context that is registered with the system while t_d is unknown until the client actually deregisters. Additionally, a context can obviously only be deregistered after having been registered first, i.e., $t_r < t_d$.

We further assume that the time on all nodes is synchronized using a method such as the Network Time Protocol (NTP) [MMBK10]. This ensures that all nodes have a sufficiently precise clock to evaluate the time constraints of messages; in particular, [Min99] finds that NTP nodes on the Internet are synchronized to within a mean offset of 8.2 ms, with a standard deviation of 18 ms. If an application requires a higher precision than that of NTP, the best-effort nature of Contextcast in all likelihood does not satisfy the application's requirements either.

Temporal Constraints for Contextcast

In addition to the temporal extents of client contexts, a temporal Contextcast requires suitable addressing predicates to restrict the set of recipients according to a temporal relation. Picking up the example from Section 6.1, a temporal Contextcast message might address "all clients who pass through the shopping mall between January 14th, 9:00, and January 28th, 17:00, and whose age is below 30". In addition to the constraints for *location* and *age*, this message also contains a temporal constraint, "between January 14th, 9:00, and January 28th, 17:00" Thus, only contexts that match the temporal restriction in addition to the other two match this message.

The concept of time and the temporal relation between events has long been studied in various fields such as database management systems or artificial intelligence. One of the most influential approaches has been the work of Allen [All83]: Typically used statements about temporal relations are both relative and contain some amount of uncertainty: "He discovered that his wallet was missing *after he got home*" or "The accident happened yesterday *during the afternoon rush hour*". Allen therefore introduces a temporal representation based on intervals and a set of 13 temporal relations. These relations are the symmetrical *equals (=)* as well as the relations *before* (<), *meets* (m), *overlaps* (o), *during* (d), *starts* (s), and *finishes* (f), together with their respective inverse (>, mi, oi, di, si, fi).

We have adapted and refined the relations from [All83] to be used as predicates for the *time* attribute in contextual message constraints. These predicates are designed to be as expressive as necessary while at the same time easy to use for developers and users of the Contextcast system. To this end, our relations differ from Allen's in the following points:

1. We have introduced an uncertainty threshold in the equals and meets relation to allow for uncertainty when specifying a temporal constraint; these two pose

the most difficulties when using exact times, but it is straight-forward to add similar uncertainty values to the other predicates should the need arise.

2. We have combined Allen's *meets* m relation and its inverse mi, since $X \,\mathrm{mi}\, Y \Leftrightarrow Y \,\mathrm{m}\, X$; we have opted to keep the before-inverse (bi) relation as a relation *after* in Contextcast, though, since it is commonly used to describe temporal relations in human interaction.

3. We have renamed Allen's overlaps and its inverse to "ends-in" and "starts-in" to make their semantics clearer.

4. Additionally, we have introduced two new relations, *overlaps* and *excludes*, to allow for an easy way to describe that a context was valid at some point during an interval or was not valid during that interval, respectively, without having to combine multiple temporal constraints with logical operations.

5. As per a note in Allen's original work, we have collapsed Allen's relations d, s, f into a relation *during* (Allen's dur) and the relations di, si, fi into a relation *contains* (Allen's con).

This results in the 10 predicates for the *time* attribute listed in Table 6.1. In addition to the predicates and their syntax, the table also shows the corresponding Allen relation as well as the semantics of these predicates, i.e., the respective comparisons of points in time. ContextRouters use these comparisons internally to evaluate the *time* constraints. Please note the differing syntax and semantics for the *meets* predicate, which depends on its use as either Allen's m or mi relation.

The predicates are generally self-explanatory, however, we are discussing a number of examples in more detail, namely the predicates *before*, *equals*, *overlaps*, and *during*:

- The predicate *before* ($time < [a, b]$) specifies that a client context had to be both registered and deregistered before the interval $[a, b]$. It is thus equivalent to:

$$time < [a, b] \quad \Leftrightarrow \quad [t_\mathrm{r}, t_\mathrm{d}] < [a, b] \quad \Leftrightarrow \quad t_\mathrm{d} \leq a.$$

- Another example is the predicate *equals*, which is a little more complicated compared to the others. It tests two intervals $[t_\mathrm{r}, t_\mathrm{d}]$ and $[a, b]$ for equality. However, since usually no two time intervals are exactly the same, the comparison contains an additional tolerance, z, for applications to specify. This is used for a fuzzy comparison on interval boundaries, i.e.,

$$[t_\mathrm{r}, t_\mathrm{d}] \approx_z [a, b] \quad \Leftrightarrow \quad |t_\mathrm{r} - a| \leq z \wedge |t_\mathrm{d} - b| < z.$$

  In other words, two intervals are considered equal if their start points and end points are each less than the time z apart.

Predicate	Syntax	Allen relation	Evaluation
before:	$time < [a, b]$	$<$	$\Leftrightarrow t_d \leq a$
after:	$time > [a, b]$	$>$	$\Leftrightarrow b < t_r$
equals:	$time \approx_z [a, b]$	$=$	$\Leftrightarrow \lvert t_r - a \rvert \leq z \wedge \lvert t_d - b \rvert < z$
meets:	$time \parallel_z [a, b]$	m	$\Leftrightarrow \lvert t_d - a \rvert < z$
	$[a, b] \parallel_z time$	mi	$\Leftrightarrow \lvert b - t_r \rvert \leq z$
ends-in:	$time \in^{ei} [a, b]$	o	$\Leftrightarrow t_r < a \wedge a < t_d \wedge t_d \leq b$
starts-in:	$time \in^{si} [a, b]$	oi	$\Leftrightarrow a \leq t_r \wedge t_r \leq b \wedge b < t_d$
overlaps:	$time \cap [a, b]$	–	$\Leftrightarrow t_r \leq b \wedge a < t_d$
excludes:	$time \neq [a, b]$	–	$\Leftrightarrow t_d \leq a \vee b < t_r$
during:	$time \subseteq [a, b]$	dur	$\Leftrightarrow a \leq t_r \wedge t_d \leq b$
contains:	$time \supseteq [a, b]$	con	$\Leftrightarrow t_r \leq a \wedge b < t_d$

Table 6.1: Temporal predicates and their evaluation by Contextcast

- As we mentioned before, the newly introduced *overlaps* predicate can be used to test whether a context was valid at any point within an interval. This corresponds to
$$[t_\mathrm{r}, t_\mathrm{d}] \cap [a, b] \quad \Leftrightarrow \quad t_r \leq b \wedge a < t_d.$$
In other words, the context must have been registered before or at the end as well as deregistered after the beginning of $[a, b]$. This way, the two intervals have at least one point in common.

- The last example is the related predicate *during* to test whether a given context was valid during a time that lies completely within the interval $[a, b]$. It is equivalent to
$$[t_\mathrm{r}, t_\mathrm{d}] \subseteq [a, b] \quad \Leftrightarrow \quad a \leq t_\mathrm{r} \wedge t_\mathrm{d} \leq b,$$
i.e., both registration and deregistration must be within the bounds of $[a, b]$.

Please note that the two predicates, *after* and *excludes*, both match contexts that are registered after a certain point in time. In particular, this allows for new matching contexts to be registered arbitrarily far in the future. Messages with these temporal constraints would therefore accumulate in the system. As we are discussing in Section 6.4.2, we recommend that administrators limit the lifetime of such messages or disallow the use of these predicates in their system altogether.

6.3.2 Historical Messages

In broadest terms, to route a historical message the Contextcast system needs to (1) be able to lookup historical contexts and determine those matching the constraints of a given message, (2) have a mechanism to determine the actual user for a matching historical context, and (3) deliver the message to the users determined from the first two steps.

Coming back to the example of the minutes of a meeting (cf. Section 6.1) with constraints on the *location* and the *time* of the meeting, the routing takes place as follows: First, the system determines all the contexts that fulfill the *location* and *time* constraints of the message. These contexts represent the participants of the meeting. Second, it must determine the actual clients of the system that these contexts are associated with. Third, and finally, they deliver the message to the clients: Either directly if they are currently connected to the system; or indirectly via the client's mailbox if they are not.

From this broad description, we can see a number of challenges for the routing of historical messages: The system must have a method to archive historical contexts accompanied by an efficient lookup to determine the historical contexts that match a given message. Additionally, the system must have the means to resolve archived contexts to clients while still protecting the privacy of the users. And finally, it needs an efficient mechanism to deliver a message to a large number of matching recipients.

In the following sections, we discuss these individual items in detail and present our approach for each. Together, they provide an efficient, privacy-aware routing mechanism for historical Contextcast messages. We then use this as the basis for a number of improvements, which help to further reduce network load from historical message routing.

Archiving Historical Contexts

As mentioned before, disseminating a historical Contextcast message requires determining the set of historical contexts that match the constraints of a given message. This in turn requires an archive of historical context data, from which to lookup the matching ones.

There are two natural choices for the archival of historical client contexts: The first possibility is a separate archive, e.g., in a (classical) database management system. Thus, the lookup of matching contexts benefits from extensive research in this field. However, because of the frequency of Contextcast messages and context updates in particular, for scalability this database management system would have to employ powerful, expensive hardware or follow a divide-and-conquer approach

such as a distributed database management system [ÖV11]. The second option is an archive of context information integrated into the Contextcast system in a distributed manner. This method is similar to a distributed database management system, yet its advantages are a finer control over the actual distribution of the data and the ability to integrate the context lookup into the dissemination of historical messages. We therefore propose a context archive that is integrated in the Contextcast overlay network, with full control over the distribution of historical contexts and the necessary lookup algorithms to determine matching recipients for historical messages.

The organization of the context archive, i.e., how contexts are distributed across the various nodes, directly affects both the effort to lookup matching contexts and the effort to archive contexts. Contextcast's design with ContextNodes' service areas and clients with mobile devices also has a number of important implications for the archival location of historical contexts: First, contexts are registered and originate in the access networks; the ContextNodes therefore have access to a pristine version of all locally registered contexts, before any aggregation (cf. Chapter 4), and independent of whether it is propagated in the network or not because of an adaptive propagation approach (cf. Chapter 5). Second, and related to the first point, a context is available directly in the access network, without additional communication. Archiving a context somewhere else in the network occurs additional overhead, both at registration and deregistration. Third, the service areas of the contexts nodes partition the network spatially and serve as a readily available spatial index. The system can employ this information to efficiently route messages to the locations where potentially matching contexts are archived, which we are discussing in more detail in the following section.

Based on these points and the resulting advantages, we propose a spatially partitioned distributed context archive for Contextcast:

Definition 6.2 (Spatially Partitioned Context Archival)**.** For historical message dissemination, a ContextNode archives all contexts that clients register in its service area, together with the time interval $\left[t_{\text{register}}, t_{\text{deregister}}\right)$ for each context.

Besides the mentioned advantages of a spatially partitioned archive, it is also possible to further enhance the context lookup with additional indexes. These could be used, e.g., when a message contains a *location* constraint with a huge area or none at all. We are discussing this option and its overhead further in Section 6.3.2. Also, archiving contexts locally at the ContextNodes requires additional storage, which was not necessary in the previous approaches, thus increasing the hardware requirements for ContextNodes.

Based on the model of time-stamped contexts in combination with a spatially partitioned archive of contexts, the question of archival time also has a straight-

forward answer: The system learns of a new context the moment a client registers it (t_{register}). The context is then available in the access network and throughout the Contextcast network as a currently registered context until deregistration. After deregistration at time $t_{\text{deregister}}$, the context must be available in the archive for historical message dissemination. Thus, any context must be archived at some point during $\left[t_{\text{register}}, t_{\text{deregister}}\right]$. Archiving the contexts when $t_{\text{deregister}}$ is known provides the lowest overhead: Any earlier archival requires an additional update when $t_{\text{deregister}}$ becomes known. Also, since archiving a context in the same access network where it was registered is practically instantaneous, archiving at time $t_{\text{deregister}}$ does not introduce any adverse delays before the context is available for historical message dissemination.

Note, though, that adding a context to an index for an attribute other than *location* is more complex: Distributed indexes for these attributes rarely place a user's context information on the same node where the client connects to the system because of their *location*. Since adding a context to the index in this case requires transmitting information to the responsible node, there is a certain delay before the information is available at that node. If the system uses the index to lookup contexts during this delay, it may miss a context, thus causing false negatives. It is therefore advisable to add context information to such an index some time before $t_{\text{deregister}}$ to ensure that the information is available at $t_{\text{deregister}}$, when the system can no longer rely on currently registered information. This requires at least an additional message from the ContextNode at $t_{\text{deregister}}$ to update the *time* attribute. Since the update message is needed in this case anyway, the system may as well add a context to another index as early as t_{register}.

Resolving Contexts to Clients

The next obstacle when disseminating messages to historical user contexts is the mapping from matching historical contexts to current clients so the message can be delivered. These clients may have changed their context or may not be connected to the system any longer.

Such a mapping is easily achieved with a unique user identifier (ID) included in all client contexts. The downside of such an unambiguous mapping from client contexts to entities is that it allows to create comprehensive movement and context profiles of clients, though. This would most certainly lead to a bad acceptance of the system due to privacy concerns. (Please note, these unique IDs are not necessary in the original system, as messages are simply delivered to anonymously connected clients with a matching context. Because the clients are still registered, the system has an implicit mapping from registered contexts to the connected clients that registered them.)

To improve client privacy, we envision a system of (globally unique) Virtual Identities (VIDs) for client contexts in Contextcast (compare, e.g., [WBS+05]). On their own, these VIDs cannot be used to identify a particular entity, as they provide no connection to physical entities and can frequently be changed. Clients can potentially choose a new VID every time they register a context with the system. This achieves two things: First, it makes obtaining a complete history of a single entity's context difficult since no pair of contexts with different VIDs can reliably be determined to relate to the same entity. Second, no attacker can reliably resolve an archived context's VID to the corresponding entity since there is nothing connecting the VID an entity. Unfortunately, to deliver historical messages to clients that are offline or have changed their VID it is still necessary to have such a mapping from VIDs to entities in the system.

Therefore, in Contextcast, we rely on trusted third parties, called Trusted Nodes (TNs), to create virtual identities for entities and resolve them for message delivery. A TN can be a client's cell phone provider or ISP, which are usually trusted implicitly, or even an independent service offering such VID management. The TNs only store a mapping VID $\mapsto$ entity, they do not need to know a client's context. With this mapping, the system can resolve a client from the VID in a context that matches a particular message and then deliver this message to that client.

When resolving VIDs to entities, the system needs to determine the TN responsible for a given VID. This can be achieved, e.g., by encoding the TN in the VID itself: [timestamp]@[Fully Qualified Domain Name (FQDN) of the TN]. The timestamps, which each TN hands out, are strictly monotonically increasing. The combination of a timestamp with the TN's FQDN ensures that VIDs are globally unique without giving away details about the actual entities. Additionally, using timestamps instead of simpler sequence numbers prevents leaking activity information from individual TNs, such as a long time between the occurrence of two subsequent VIDs.

Basic Historical Message Routing

As mentioned before, routing a historical message consists of three conceptual phases: (1) the Context Lookup, in which the system determines the historical contexts that match the given message, (2) the VID Resolution, in which the system resolves the historical contexts to the actual clients of the system, and (3) the actual Message Delivery, in which the message is routed to the clients whose historical context matches the message. In the following paragraphs, we detail each of these phases.

Context Lookup. During the Context Lookup phase, the ContextRouters employ an index for one or more attributes to route a message to all the access networks where potentially matching contexts are archived. Because of the reasons given

in Section 6.3.2 and its status as *primary context* [RBB03], we show an approach in Contextcast that uses the *location* attribute for this first phase. The benefit is that every context possesses a *location* (where it was registered) and that the ContextNodes' service areas serve as a readily available index.

Routing a message according to a target *location* is in essence a geocast (compare, e.g., Section 2.3 or [Dür10]). To forward messages to the access networks, each router needs information about the geographical coverage of the other nodes in the network, i.e., the information what locations can be reached via each of its link. This can be achieved in several ways, e.g., via a special context that each ContextNode propagates, which contains its own service area and which allows other routers to compute shortest paths and bounding areas for its neighbor routers.

In the following algorithms, `ServiceArea`(N) denotes the service area of a ContextNode N, while `ServiceArea`(l) denotes the cumulative area that can be reached via a link l. Algorithm 6.1 shows the pseudo code for this phase. When routing a historical message, each router first checks whether this message was received and thus processed before. While this does not happen in an acyclic network, such a test allows an extension to arbitrary topologies in the future. Then it forwards the message via each link for which the service area of the link intersects with the message's target location.

If a router is also responsible for an access network, i.e., a ContextNode, and its service area overlaps with M's target *location*, it continues with determining the locally archived, matching contexts. To this end, it determines for each context whether it matches the constraints and then adds the VID to the result set accordingly. The set eliminates all obvious duplicate VIDs, which can happen when a context is registered several times in one access network, e.g., due to repeated disconnects of its wireless connection.

There exist efficient methods for the local storage of multi-attribute data in a database management system and retrieve those records matching a certain set of constraints. Anyone operating a ContextNode can choose from any number of approaches to store and lookup context information. However, the efficiency of local database lookups is not the focus of this dissertation, the interested reader is referred to [GMUW08, KE13].

Virtual Identity Resolution. During the second phase, the VID Resolution phase, the system determines the actual client associated with each historical, matching context. The ContextNodes that have archived contexts matching M send copies of the message to the Trusted Nodes responsible for one of the matching contexts' VIDs from the previous phase. The pseudo code for this step is shown in Algorithm 6.2. It creates a set of all TNs that are responsible for subsets of the given VIDs. For each of these TNs, the ContextNode creates a copy M' of M, tags it with the corresponding subset of VIDs that the TN is responsible for, and forwards M' to this TN. Thus, a

Algorithm 6.1 Context Lookup

Require: A ContextRouter N and a historical Contextcast message M.
Ensure: M forwarded over all links whose cumulative service area intersects with M's target location and mVID the set of VIDs of the locally stored contexts that match M, i.e., mVID = {Set of VIDs of locally archived contexts : $C_{\text{VID}} \sqsubset M$}.

```
if M was not received before then          ▷ Does not happen in acyclic networks.
    for all l ∈ {links of N} do
        if ServiceArea(l) ∩ location_M ≠ ∅ then    ▷ Intersects with target location
            forward M via l
        end if
    end for
    if (N is a ContextNode) ∧ (ServiceArea(N) ∩ location_M ≠ ∅) then
        mVID ← ∅
        for all C ∈ {locally stored contexts} do          ▷ Naive iteration.
            if C ⊏ M then
                mVID ← mVID ∪ (VID of C)
            end if
        end for
    end if
end if
```

single message is sent from a given ContextNode to each of the TNs responsible for one or more VIDs of matching contexts.

Algorithm 6.2 VID Resolution

Require: A historical Contextcast message M and a set mVID of the VIDs of contexts matching M.
Ensure: M sent to the Trusted Nodes for all VIDs in mVID.

```
TNs ← ∅                                  ▷ The TNs responsible for one or more VIDs
for all VID ∈ mVID do
    TNs ← TNs ∪ Trusted Node for VID
end for
for all TN ∈ TNs do
    Create a copy M′ of M
    Tag M′ with VIDs that TN can resolve
    Send M′ to TN
end for
```

While the algorithm resolves the VIDs from phase one, it has its shortcomings: For instance, it sends a copy of the message M as a unicast message to each of the TNs, which causes a high network load if clients employ many different TNs. One way to reduce the number of these messages is a multicast approach, which duplicates message in the network as needed. This is not a trivial task, though, as the group of TNs depends on the set of matching contexts and the resulting VID set, which in turn would require a vast number of multicast groups. Additionally, due to their movement, a client may have their matching context archived in multiple access networks, thus causing the same VID to be resolved from several ContextNodes. To reduce this effect, we are showing an optimized variant that can suppress such duplicate VIDs resolutions that result from the clients' mobility.

Historical Message Delivery. Once a message reaches a Trusted Node, the TN can lookup the associated VIDs it contains and deliver the message to the corresponding clients; Algorithm 6.3 shows this message delivery. It starts by looking up the entity represented by a given VID. If the corresponding client is currently connected to the system (under a potentially different VID registered with this TN), the TN can deliver the message directly to that client. If the client is not connected to the system or the TN is not aware of it, for instance because the client is using a different TN, the message is delivered to the client's mailbox. We assume that clients poll their mailbox in regular intervals.

In the absence of failures, the algorithm delivers a message to a single client at most once from one TN. This even works if several VIDs of a physical entity may have matched M, as long as all those VIDs were registered with the same TN. If a client

Algorithm 6.3 Historical Message Delivery

Require: A historical message M tagged with a set mVID of matching contexts' VIDs.
Ensure: M delivered to all matching recipients.

```
for all VID ∈ mVID do
    R ← physical entity corresponding to VID          ▷ Recipient R
    if not R has already received M then
        if R is currently connected then
            Deliver M directly to R.
        else
            Deliver M to R's mailbox.
        end if
    end if
end for
```

obtains multiple VIDs from different TNs and then registers contexts with these VIDs, they may receive a message multiple times: To ensure the privacy properties of using VIDs as pseudonyms for physical entities, TNs must not cooperate. Thus, if multiple contexts from one client match a message and different TNs are responsible for delivery, each of these TNs needs to deliver the message; they do not know about the other TNs or other matching contexts.

Optimized Historical Message Routing

The algorithms in the previous paragraphs served as an introduction of the general concepts of historical message routing. They have a few shortcomings, though, which we are going to address in the following paragraphs.

Optimized Context Lookup with Duplicate VID and TN Removal. As client contexts generally represent mobile users, a client can move from one access network to another one at any time. This results in having the same context archived in several access networks. Since each ContextNode looks up matching contexts locally, independent of other nodes, during the Context Lookup phase, a message with a target location spanning the service areas of multiple ContextNodes may match the same context in each of these access networks. The previously shown algorithm—which we from now on call the Simple Historical (SH) algorithm—starts the VID Resolution directly with the local lookup results. The same VID is therefore resolved from all these access networks, without consolidating duplicate VIDs from different access networks. As a result, several ContextNodes send a given message to the same TN for the same matching context.

In addition to the duplicate VID resolution resulting from single contexts archived at several nodes, also contexts from different clients that use the same TN cause overhead: While the there may exist a great many number of TNs, in reality we expect many clients to rely on third-party operated TNs that are provided, e.g., by their ISP. Thus, the set of VIDs of the contexts matching a given message may need to be resolved by only a few well-known TNs. If these VIDs are the results from different access networks, starting the VID resolution from each one (the SH algorithm) causes a message to be sent to the same TN multiple times from different access nodes.

To avoid these two types of overhead, we introduce the Fully Consolidating Historical (FCH) algorithm: it collects the matching VIDs for a given message from the access networks and consolidates them by removing obvious duplicates, i.e., identical VIDs. After this step, the algorithm starts the VID Resolution from the node where the results were collected, removing duplicate TNs in the process.

To this end, we record the distribution tree for the messages so the results can be returned along the same tree: Every time a ContextRouter forwards a message to more than one neighbor (i.e., children in the distribution tree) or to a single neighbor but needs to perform a local lookup, it records its own ID in the message. This sequence marks all the branching points in the distribution tree down to the leaves, i.e., all the nodes where results from subtrees or local lookups can be collected. See Figure 6.3 for an example of a message and its distribution tree, with branching points recorded in the message.

With the information about the distribution tree, leaf nodes can return the results from locally archived contexts to the previous node where the distribution tree branched. This router then consolidates the results, i.e., unifies the sets of VIDs from all its subtrees with its local results, and sends the result back to the previous branching point in the tree. Again, this router consolidates the VID sets and so on until the results reach the first branching point (node A in Figure 6.3). At this point, the node has the consolidated results from all access networks with potentially matching contexts. This node can then start the VID Resolution, sending the message to the set of unique TNs that are responsible for the matching VIDs. (For succinctness, we omitted the discussion of lost results from subtrees. Obviously, the system needs a mechanism, e.g., timeout and/or retransmit, to prevent these situations.)

Between these two extremes SH and FCH, one can also imagine variations that collect the results but limit the level up to which to collect the results. We call these algorithms Partially Consolidating Historical (PCH). The nodes that collect results from their subtrees are called "consolidation points". An additional number $n = 1, 2, 3, \ldots$ denotes the consolidation point where we collect the results. Thus, PCH1 is identical to the FCH algorithm, as the consolidation point is the first

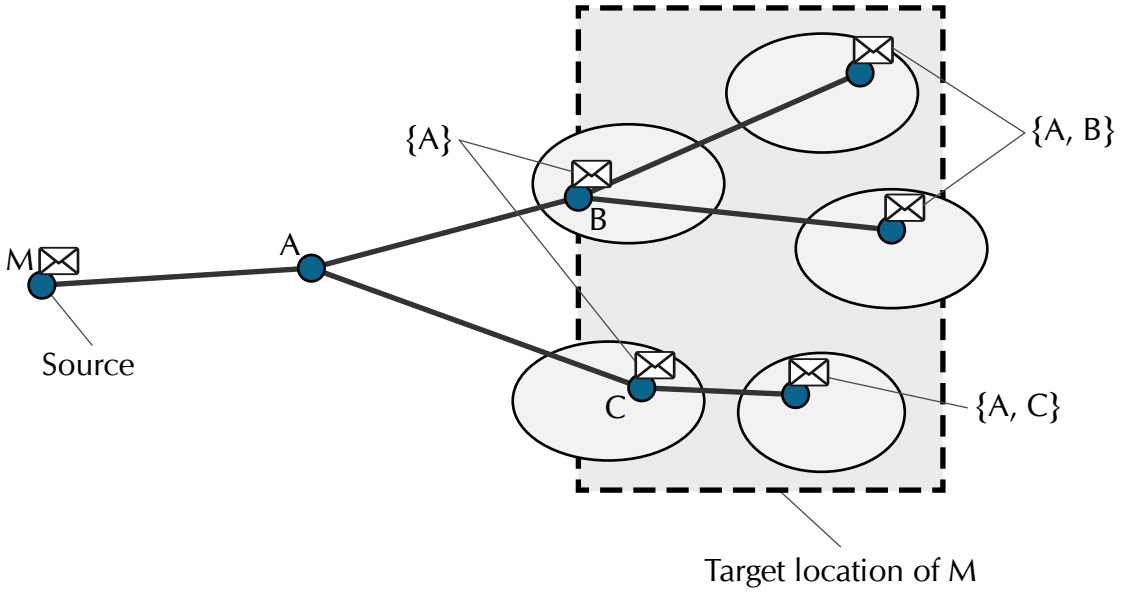


Figure 6.3: A historical message M's distribution tree, with branching points recorded in the message along the route

branching point. PCH2 collects the results at the second branching points (nodes B and C in Figure 6.3), PCH3 at the third (which does not exist in the example), and so on. As one can see from the example, PCH<n> usually results in several nodes being the n-th branching point in their respective subtrees. Thus, multiple nodes collect the results for their subtree and start the VID resolution, but fewer than if each ContextNodes starts it. Algorithm 6.4 shows the pseudo code to collect the results from the subtrees until we reach the intended consolidation point.

Algorithm 6.4 Optimized Context Lookup

Require: A branching point N of the Context Lookup distribution tree, a historical Contextcast message M, and a consolidation depth k.
Ensure: mVID = {Set of all VIDs : $C_{\text{VID}} \sqsubset M$} collected at consolidation point.
  mVID ← Consolidated results from all subtrees
  if the k-th branching point is an ancestor of N in the distribution tree **then**
    Send mVID to previous branching point.
  else ▷ Consolidation point reached or less than k branching points on this path.
    Start VID Resolution.
  end if

Collecting the resulting VIDs suppresses duplicates from different ContextNodes. However, it also causes an increase in end-to-end message delay because of the

time it takes to collect and reconcile the results from the different subtrees as well as an additional messages. Thus, the choice of algorithm—SH, FCH, PCH2, PCH3, ...—depends on the mobility of the clients, i.e., the amount of duplicate VIDs, the diversity of TNs, as well as the additional processing load on the nodes that is acceptable. (The additional delay introduced by this consolidation is irrelevant in practice since the addressed contexts are historical and therefore the delay is typically small compared to the age of the context.) An administrator can select a suitable algorithm based on the observed conditions in a Contextcast system.

But even with this optimization to the Context Lookup, it is impossible to avoid all duplicate messages: Since a client can change VIDs at any time to increase privacy, not all duplicate VIDs can be recognized as such. This is a disadvantage of the chosen level of privacy. But as each Trusted Node receives only one copy for all matching VIDs, this only increases the message size. The Trusted Nodes are able to discover and reconcile all the different VIDs for every physical entity.

Optimized VID Resolution using Multicast. A second drawback of the previously presented approach is the sending of unicast messages to each Trusted Node during the VID resolution. As a result, the same message is forwarded multiple times over links that the different unicast paths share. This places unnecessary additional load on the network.

One solution to reduce the load from these messages is to employ a form of multicasting (cf. Section 2.3). However, for any given message the VID set of matching recipients must be transmitted to an element of the power set of all TNs, $\mathcal{P}(\{\mathrm{TN}_i : i = 1, \ldots, n\})$. This would require establishing 2^n separate multicast groups for every combination of n TNs. If we assume that there are very few TNs in the system, this might actually be possible. But even for only 10 TNs, it requires more than 1 000 multicast groups. And with the open nature of the Nexus platform (cf. Section 1.2.3), anybody can operate a TN, even individual clients of the system. Thus, as we already argued in Section 2.3, such a large number of groups poses a scalability problem.

So, instead of relying on a network or application layer multicast mechanism with pre-populated groups, we adapt Explicit Multi-unicasts (Xcast, cf. [BFI+07]) to route such messages along the overlay network. An Xcast message contains the set of all the recipients of a message, i.e., the TNs in our case. From this, the routers construct the distribution tree on the fly, based on available unicast routing information. In the process, they duplicate the message and partition the recipient sets as needed. This scheme avoids the large amount of multicast groups at the expense of a more complex message forwarding and larger messages.

Algorithm 6.5 shows how a ContextRouter—the consolidation point from the previous section—determines the TNs that are responsible for the VIDs resulting from the Context Lookup. After that, it tags M with the VIDs that matched this

message and the corresponding TNs and sends it via an explicit multi-unicast to the TNs.

Algorithm 6.5 Multicast VID Resolution

Require: A historical Contextcast message M, a set of matching VIDs mVID collected from subtree.
Ensure: M sent to all Trusted Nodes responsible for all VIDs that matched M.
TNs $\leftarrow \emptyset$
for all VID $\in$ mVID **do**
TNs $\leftarrow$ TNs $\cup$ Trusted Node responsible for VID
end for
Tag M with sets TNs and mVID
Send M via an *explicit multi-unicast* to all elements in TNs

Employing explicit multi-unicasting instead of multiple unicasts for VID resolution allows ContextRouters to exploit shared links between the various unicast paths. The routers partition the recipient set, i.e., the set of TNs, according to the unicast next hop information for each TN. Algorithm 6.6 shows the necessary steps. For each neighbor, a router tests whether this neighbor is the next hop for a subset of TNs. If it is, it forwards a copy of the message, tagged with the subset of TNs and the subset of VIDs that these TNs can resolve to this neighbor.

Algorithm 6.6 Explicit Multi-unicast Forwarding

Require: A VID resolution multicast message M with the recipient set TNs and the set of matching VIDs mVID, e.g., created by Algorithm 6.5.
Ensure: M forwarded to the next hops towards all TNs listed in M.
for all N $\in$ `Neighbors` **do**
Reachable $\leftarrow$ {TN $\in$ TNs : N is next hop for TN}
TNs $\leftarrow$ TNs $\setminus$ Reachable ▷ Remove those we reach via N
HandledVIDs $\leftarrow$ {VID $\in$ mVID : VID can be resolved by TN $\in$ Reachable}
mVID $\leftarrow$ mVID $\setminus$ HandledVIDs ▷ Remove those we handle in this loop
if HandledVIDs $\neq \emptyset$ **then**
Create a copy M' of M without the original sets TNs and mVID
Tag M' with recipient set Reachable and matching VID set HandledVIDs
Forward M' to N
end if
end for

Additional Attribute Indexes. During the Context Lookup, Contextcast determines matching contexts, which are stored on the ContextNodes. To this end, it

employs a spatial index created by the ContextNodes' service areas and a message's target *location*. The reasons behind this design choice are scalability and simplicity:

1. Contexts are stored distributedly, so no single node becomes a bottleneck during the Context Lookup.

2. Storing contexts in the service area where they were initially registered requires no communication overhead to transmit the information to a different storage node.

The approach has a drawback, though: It requires all messages to have a target *location*. Without it, the Context Lookup would require a broadcast to all service areas to lookup matching recipients. Due to *location*'s importance as *primary context*, we consider this a minor limitation. However, even if a *location* constraint is enforced for temporal messages, it may cover a large area, thus requiring many messages to lookup matching contexts in the targeted service areas. In the extreme case, the *location* might be so large that the lookup effectively becomes a broadcast.

Contextcast can incorporate distributed indexes for context attributes other than *location* to complement the existing spatial index in the routing of historical messages: For an index on an additional attribute α_i, the system maps α_i's value range to a set of nodes, e.g., by means of a Distributed Hash Table (DHT) such as Chord [SMLN$^+$01], Pastry [RD01], Tapestry [ZKJ01], or CAN [RFH$^+$01]. The nodes then store a copy of the contexts whose value of α_i are within their respective value range. (In principle, the nodes need not store the actual context data, instead they could simply store a reference to a node storing the data. Because of the relatively small size of a client context, the savings would be negligible. However, it would prevent the Contextcast system from locally evaluating all constraints of a message against a stored context, adding another indirection and further increasing the message load in the Context Lookup.)

During the Context Lookup the system can use such an index to find matching contexts for the corresponding constraint, e.g., when the *location* constraint is unsuitably large or not present. Similarly to using *location* as described in Section 6.3.2, the system routes a historical message to the nodes who are responsible for values that would match the constraint on α_i. These nodes then evaluate their stored contexts against the message constraints to determine the set of matching recipients.

While such additional indexes can be used instead of the spatial index in the Context Lookup, their maintenance, i.e., updating contexts or adding new ones, generates overhead as well. Storing—as well as retrieving—an item in a DHT requires routing to the node responsible for the corresponding key. Therefore, any update of a context with an attribute α_i means routing the update to the node responsible for this particular value of α_i. In typical DHTs (e.g. Chord [SMLN$^+$01]

or Pastry [RD01]), the route length and thus the number of messages is $\in \mathcal{O}(\log N)$, with N the number of nodes in the network. (There are algorithms that achieve shorter routes at the cost of a higher node degree, e.g., CAN [RFH+01].) The reference algorithm (see Section 3.3.3) broadcasts all contexts in the network, in particular if a context contains α_i, the information also reaches the node responsible for this context's value of α_i. Additional attribute indexes therefore would not incur any overhead when a system used the reference algorithm. However, with the optimizations we have introduced in Chapter 4 and Chapter 5, the system to no longer propagates all contexts in their pristine form through the network: They may get aggregated with other contexts or only forwarded when there is a sufficient amount of false positives. Since it cannot rely on other propagation mechanisms, an additional distributed index for α_i causes a $\mathcal{O}(\log N)$ message overhead for each context update with this attribute in the system. In contrast, the spatial index on the contexts' *location* does not suffer such an overhead. Every context is stored on the node where the client connected to the system due to their physical presence in the service area. This requires no additional messages for maintaining a spatially partitioned context store and the communication overhead is therefore $\in \mathcal{O}(1)$.

Despite the different maintenance overhead, we expect the spatial index and any DHT-based distributed attribute index to behave similarly in the Context Lookup. In both cases, the system routes a historical message to the nodes responsible for values that satisfy the constraint for the corresponding attribute, duplicating messages as necessary to form a distribution tree. This distribution tree, especially its size, i.e., width and depth, depends on a number of aspects:

1. The selectivity of the actual constraint and consequently the number of nodes in the index that may have matching contexts; these are the leaves in the distribution tree.

2. How well an index preserves locality, i.e., whether similar values are stored on the same node or closely connected ones, which is obviously a strong point of the spatial index.

3. The average route length in the index, which determines the depth of the distribution tree and which depends on the degree of the participating nodes.

Depending on these factors, especially the selectivity, the spatial index may work well for the Context Lookup or it might be preferable to employ other attribute indexes. If historical messages are relevant only in smaller areas and this fact is reflected in their *location* constraint, the spatial index is a natural choice for the Context Lookup. Should the spatial index prove inadequate, Contextcast can easily be extended with additional indexes as outlined in this section. However, additional

indexes must incorporate the other two factors, i.e., the preservation of locality and a low average route length. While typical DHTs achieve the latter one with an average route length of $\mathcal{O}(\log(N))$, the selected mapping of attribute values to nodes needs to account for the locality of values.

In fact, one can even imagine an adaptive mechanism similar to the one we have shown in Chapter 5, which would observe historical messages and the performance of the spatial index. Such an approach would compare other attributes that are commonly used in historical messages, estimate the effort required to maintain a distributed index for each such attribute, and compare the effort to the potential message savings for this index when used instead of or in addition to the spatial one. Based on the result, the system can set up additional indexes to be used for the Context Lookup; for each historical message, it can then determine a suitable index based on the constraints used in the message and their respective selectivity.

6.3.3 Future Messages

By their definition, future messages are delivered to matching contexts *after* they have been sent. In some ways, this makes the dissemination of future messages very similar to the approaches we have shown in the previous chapters: After a future message has been sent, the system can match new contexts against it continuously, as if the message was sent just at that very moment. This removes the need to lookup historical contexts and the added complexity of VIDs; if a client is still connected to the system when their context matches a message, it can be delivered directly. Thus, future messages are conceptually simpler than historical ones. A few of the predicates, however, depend on the time a context is deregistered for evaluation (cf. Table 6.1). In this case, a VID resolution is necessary. And if the user is no longer connected, e.g., with a newer context, it also requires a delivery via the client's mailbox.

Obviously, the Contextcast system cannot know in advance for a given future message if or where in the network a client's context is going to match the message's constraints. Disseminating messages in advance and storing copies throughout the network or even in every access network has the advantage of low latencies when a match occurs; but it generates false positives if no client registers a context at some time in the future, which matches the temporal and other constraints of the message. In the next section, we therefore present a reactive routing approach to prevent such false positives for future messages.

Reactive Forwarding of Future Messages

As stated before, forwarding messages that never find a matching recipient places unnecessary, false positive load on a Contextcast system. To avoid this, Contextcast employs a reactive forwarding strategy for future messages: A message is recorded and only forwarded once a matching context is actually registered with the system.

This is possible due to a particular property of the Contextcast system (compare Section 3.3.3): ContextRouters propagate information about all newly registered and updated contexts as well as deregistered contexts in the network. Thus, if a context C matches a message M ($C \sqsubset M$) sent at time t, C either was already propagated throughout the system at time t, or it is propagated throughout the system some time after t. (Please note that such a context C may be propagated either as is or, as we discussed in Chapter 4, as part of an aggregation C'. However, any aggregation that fulfills the aggregation condition—see Definition 4.3—guarantees that $C \sqsubset M \Rightarrow C' \sqsubset M$.) This particular property allow us to store messages very close to message senders, reacting to future contexts when clients register/deregister.

This recording of messages could even take place directly on the mobile device of a sender. Thus, a message would never leave a client's device if no matching recipient ever registers a context, causing no load at all. However, this would, prevent a sender from disconnecting from the network, for instance, to save energy. Therefore, in our approach, the first infrastructure node that a client connects to stores a message, i.e., typically a ContextNode.

Algorithm 6.7 shows the recording and reactive routing for future messages. For a new message, the ContextRouter determines for each neighbor whether one of the *current* contexts matches the message. If this is the case, it tags the message with the matching VIDs and forwards it accordingly. (Tagging the message with matched VIDs speeds up the matching and forwarding process: routers can skip the matching if the message has to be forwarded to a neighbor because of one of these already matched recipients.) If there is none and thus the message is not forwarded right away or if there may be future clients in its own service area, the node has to record it for potentially matching contexts in the future. These additional copies in the network avoid repeatedly forwarding the same message from the node with the original copy of the message.

Besides new future messages, whenever a ContextRouter receives a context registration or a deregistration, it must evaluate the context against the recorded future messages; this is shown in Algorithm 6.8. They forward a message if (1) the new context matches the message's constraints, and (2) it has not been forwarded over that particular link before. This reactive forwarding is a trade-off between unnecessary forwarding and increased delay for the first recipient when a matching client finally registers.

Algorithm 6.7 Future Message Forwarding and Storing

Require: A future Contextcast message M, received from router N_M.
Ensure: M forwarded to all neighbors with currently matching contexts and possibly stored locally.

```
if M was not received previously then
    children ← ∅
    for all N ∈ Neighbors \ N_M do
        if ∃C ∈ {contexts reachable via N}: C ⊏ M then
            Create a copy M′ of M
            Tag M′ with the VIDs of matching contexts
            Forward M′ to N
        else if (location ∉ M) ∨ (ServiceArea(N) ∩ location_M ≠ ∅) then
            children ← children ∪ N          ▷ Potential recipients in the future
        end if
    end for
    if children ≠ ∅ ∨ ServiceArea_local ∩ location_M ≠ ∅ then
        Store M for future contexts
    end if
end if
```

Algorithm 6.8 Reactive Forwarding of Future Messages

Require: A registration or deregistration of a context C, received from router N_C.
Ensure: $\forall M : C \sqsubset M \Rightarrow M$ forwarded to N_C or C is recorded for later delivery.

```
if C is propagated due to a deregistration then
    for all M ∈ {stored messages : depending on t_d ∧ not forwarded to N_C} do
        if C ⊏ M then
            Collect VID of C for delivery later          ▷ compare Section 6.3.2
        end if
    end for
else                                                     ▷ Registration
    for all M ∈ {stored messages : independent of t_d ∧ not forwarded to N_C} do
        Create a copy M′ of M
        Tag M′ with the VID of C
        Forward M′ to N_C
    end for
end if
```

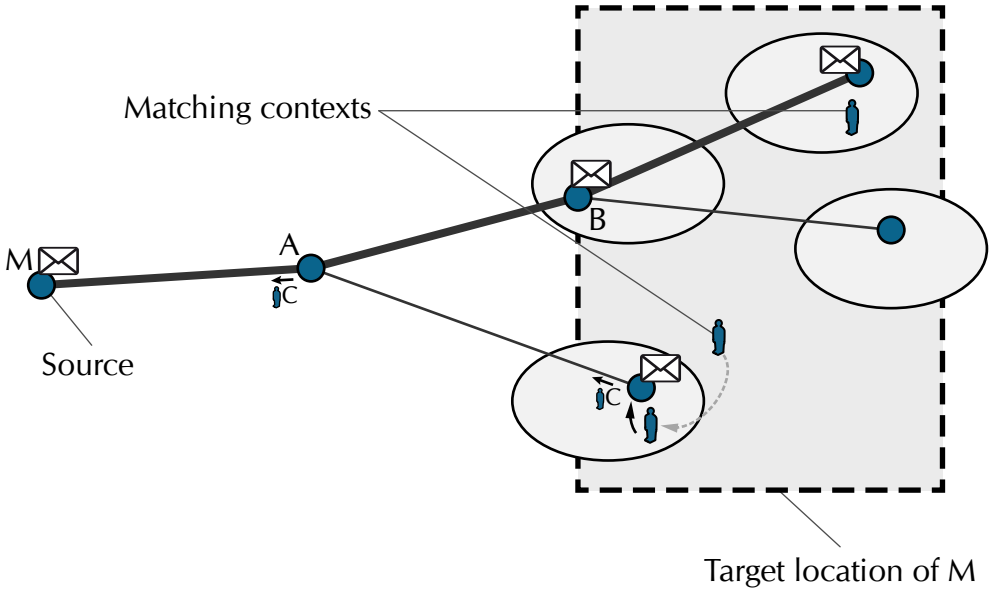


Figure 6.4: Reactive forwarding of future messages: Initially stored at the source; at B and in two access networks after the matching contexts are registered

Figure 6.4 shows an example for this approach and the resulting distribution tree for a given message. The message is first stored by the sender directly (or rather the sender's access node). Once the two matching contexts are registered and propagated through the network (shown for one of the contexts by the arrows in the figure), the reactive algorithm forwards the message to the two access network. There, they are delivered to the two clients as well as stored in each access network for further recipients in the future. Node A on the delivery tree has already forwarded it to all nodes where potential matching clients could register and thus does not need to store a copy of the message. Node B, however, has a neighbor where matching clients could register in the future and has not forwarded it to this neighbor, thus it stores the message as well. Because of this, whenever a new context or an update of an existing one gets propagated in the network, B must also match the context against the stored message(s) and forward the ones the new context matches.

Expiration of Future Messages

A reactive approach requires additional storage space for future messages. The system does not need to store messages indefinitely, though; it can remove stored messages when no newer contexts can match the message. An example for this is a message that is addressed to some contexts *before* an interval $[a, b]$ in the future.

Predicate	Syntax	Expiration
before:	$time < [a,b]$	$t_{\text{now}} > a$
after:	$time > [a,b]$	—
equals:	$time \approx_z [a,b]$	$(t_{\text{now}} \geq (b+z))$ $\vee ((t_{\text{now}} > (a+z)) \wedge (\forall c \in \mathrm{C} : \lvert t_{\text{r}} - a \rvert > z))$
meets:	$time \parallel_z [a,b]$	$t_{\text{now}} \geq (a+z)$
	$[a,b] \parallel_z time$	$t_{\text{now}} > (b+z)$
ends-in:	$time \in^{\text{ei}} [a,b]$	$(t_{\text{now}} > b) \vee ((t_{\text{now}} \geq a) \wedge (\forall c \in \mathrm{C} : t_{\text{r}} > a))$
starts-in:	$time \in^{\text{si}} [a,b]$	$t_{\text{now}} > b$
overlaps:	$time \cap [a,b]$	$t_{\text{now}} > b$
excludes:	$time \neq [a,b]$	—
during:	$time \subseteq [a,b]$	$t_{\text{now}} > b$
contains:	$time \supseteq [a,b]$	$(t_{\text{now}} \geq b) \vee ((t_{\text{now}} > a) \wedge (\forall c \in \mathrm{C} : t_{\text{r}} > a))$

Table 6.2: Expiration conditions for temporal predicates

Once the current time t_{now} is past the beginning of the interval, a, no newer contexts can match the temporal predicate. Similar conditions apply to the other predicates, some of which, in addition to the current time t_{now}, also depend on the set C of contexts that are known at that time. Table 6.2 lists these expiration conditions for the various temporal predicates.

The predicates *after* and *excludes* are slightly more challenging, though: Any contexts that are valid after the specified time match them, therefore clients matching such messages could occur arbitrarily far in the future. As a consequence, routers would have to store such messages forever. As a result, we recommend that Contextcast operators define a maximum message lifetime for these predicates or disallow their use altogether. In fact, depending on the actual values for a and b, other predicates can also cause long message lifetimes. To avoid having to store messages for years, we recommend a maximum lifetime for all future messages.

However, in a distributed system of ContextRouters, these expiration conditions are not as simple: Client contexts can take an arbitrary time to be propagated through the system. To account for this, routers cannot simply delete future messages when their expiration condition is fulfilled. Instead, they need to distinguish these three cases:

(1) It has forwarded a message to all neighboring routers where possible recipients can register *and* no local context can match the message (either because the node is not responsible for an access network or because of the message's

constraints). In this case, the router can safely delete the message, as all nodes where matching recipients might connect in the future already have a copy.

(2) A node has recorded a message for a potentially matching context in the direction of a neighboring router, but the expiration condition of the message is fulfilled. In this case, it can schedule the message for removal after a certain time. This time can be empirically deduced from the observed delay in the network; for instance, removal could take place after the time in which, e.g., 95 % of messages traverse the network, accepting a certain message loss for exceptionally high delays.

(3) While there are still neighboring routers where a matching client might connect *and* the expiration condition of a message is not fulfilled, a node keeps the recorded message to reactively propagate in the future.

Reactive Forwarding with Adaptive Propagation of Client Context Information

Reactive forwarding of future messages relies on every new context being propagated in the system. With the adaptive forwarding of client contexts, which we introduced in Chapter 5, routers no longer propagate every context, only those that improve network load by allowing routers to prune the distribution tree. In the following paragraphs, we are showing how this affects the reactive forwarding of future messages.

One implication of the adaptive context propagation is that routers need a method to distinguish between "no matching context exists" and "no matching context has been propagated to me". The solution we presented in Chapter 5 is the concept of *composite contexts*. A composite context contains a neighbor's knowledge for a certain set of attributes, i.e., from all contexts it has received with these attributes. Thus, if a router has received a composite context for a superset of the attributes used in the constraints of a particular message, it can evaluate the constraints against this composite context.

The same reasoning allows a ContextRouter to delay a message and not forward it to a neighbor immediately: as long as a node possesses a composite context with an attribute set that is a superset of a given future message (except for the attribute *time*), it will receive information about possibly matching contexts in the future.

Without the necessary composite contexts, the system must speculatively disseminate future messages, to make sure that they reach all clients despite their context not being propagated in the network. If these speculatively propagated messages never reach a matching recipient, they can be recorded as false positives, just like messages without a temporal constraint. Thus, the analysis of false positives versus update load works independent of whether a message contains a *time* constraint; if

the future messages contribute too many false positives, the system will establish a corresponding composite context, thus allowing the reactive forwarding for a certain class of future messages.

In the unlikely case that messages in the system are highly diverse, with hardly any similarities in their constraints, their forwarding due to a lack of composite contexts can cause a sizable amount of false positives. They may even exceed the update load of the client contexts. This is no different from messages without temporal constraints, though. In this extreme case, it is advisable that an administrator deactivates the adaptive propagation of contexts in the system.

6.3.4 Hybrid Messages

As mentioned in the introduction, we refer to "hybrid messages" as messages whose *time* constraint addresses both historical and future messages. An example for this is a message M sent at time $t_M = 10{:}45$ that contains a *time* constraint of $time \cap [10{:}15; 11{:}15]$. (The times in the example were shortened on purpose, as a complete date and time specification would only add to the complexity without additional insight.) This constraint matches client contexts that were registered some time during the half hour before or after the message is sent.

From this description, it becomes clear that such messages need to be treated separately as a historical and a future message: The historical part of the message needs to lookup archived client contexts, whereas the future part needs to be stored and reactively forwarded once a new, matching context is registered.

Even though Contextcast treats historical and future messages separately, there is a synergy between the two parts that can be exploited for hybrid messages. We can combine the routing of the historical part of such a message with Algorithm 6.7: when a router receives a message during the historical message routing, either during Context or VID Lookup, it also processes it as a future message; if necessary, it can store it for future clients as per Algorithm 6.7. This achieves a certain dispersal of a future message before any actual reactive forwarding. ContextRouters record the fact that they have forwarded a hybrid message during their historical message routing part, just like regular reactive forwarding of future messages.

The benefit of the approach is twofold: First, it reduces the delay of future messages as they have already been forwarded to a node closer to the recipient during the historical dissemination. Second, it saves bandwidth for the future message part, since the message has already been partially disseminated and stored on additional nodes. Routing the historical part is unavoidable, so any reuse of the results of this directly benefits the system in the reactive forwarding of the future part.

6.4 Evaluation

In the next sections, we take a closer look at the approaches to disseminate historical and future messages. In particular, we examine the message load, delay, and storage space requirements of our approaches.

6.4.1 Historical Messages

The approach for routing historical Contextcast messages we presented in Section 6.3.2 is strongly tailored to the protection of privacy of the clients while reducing the resulting overhead. To this end, contexts contain VIDs and the system suppresses duplicate messages resulting from the mobility of the clients and their ability to change VIDs at will. In the following sections, we are discussing the actual network load caused by the dissemination of historical messages as well as the resulting trade-offs: longer message delay caused by the duplicate suppression and increased message size resulting from the explicit addressing of TNs in the VID Resolution. Also, we are giving an estimate of the storage requirements for archiving historical contexts.

Setup

To examine the efficiency of our approach, we implemented a prototype with support for temporal Contextcast in the simulator PeerSim [MJ09] and use it to evaluate our algorithms. As the basis for our experiments, we set up an overlay topology of $n = 10\,000$ routers, which are uniformly distributed over a normalized area $[0,1] \times [0,1]$. The links between the routers are established using a Heuristically Optimized Trade-off [FKP02], i.e., nodes are added to the system sequentially; a new node n_i connects to an existing node n_j that minimizes the weighted sum $\gamma \cdot d_{ij} + h_j$, where d_{ij} is the Euclidean distance between n_i and n_j and h_j is the network distance to the first node that was placed. We choose the parameter $\gamma = 20$, which is less than $\sqrt{n} = \sqrt{10\,000} = 100$, and therefore, according to [FKP02], leads to a strong clustering of routers. We expect to see similar clusters in real Contextcast systems, corresponding, e.g., to local networks operated by different network providers.

From these 10 000 routers, 70 % are then selected as access nodes. Access nodes are usually closer to the edge of the network, i.e., nodes with few neighbors are more likely to be access nodes. To achieve this distribution of access nodes, we sort the nodes by their degree and use a Zipf distribution to select the ContextNodes. The probability of selecting the x-th node out of N is then given as

$$P(X = x) = \frac{x^{-s}}{\sum_{n=1}^{N} \frac{1}{n^s}}.$$

For 10 000 nodes, we selected the distribution parameter $s \approx 0.95$ to achieve a distribution where a randomly chosen node is with probability 0.8 from the first 20 % of the list. Additionally, for historical messages, 300 overlay nodes were also selected uniformly to act as Trusted Nodes.

In the following section, we show the effect of consolidating the recipient set for historical messages by comparing the message load for the various approaches we have introduced in Section 6.3.2.

Message Load

For the evaluation of our approach, we focus on the Context Lookup and VID Resolution, especially on the optimizations that suppress duplicates. We compare the different algorithms which we introduced in Section 6.3.2:

1. The Simple Historical (SH) algorithm does not suppress duplicate VIDs. It starts the VID resolution with the local results from each ContextNodes.

2. The Fully Consolidating Historical (FCH) algorithm provides the other extreme. It collects the results from all subtrees (and thus all ContextNodes inside the target *location*) at the first router where the distribution tree branched. With the complete result set, this router can easily eliminate obvious duplicate VIDs from the result set and then send only a single explicit multi-unicast message to the Trusted Nodes.

3. A number of algorithms PCH<n> ($n = 2, 3, 4$) provide different levels between these two extremes by specifying the consolidation points for the local result sets. These algorithms cannot fully eliminate duplicates and therefore several branching points send multicast messages to the TNs. (For $n = 1$ this degenerates to the FCH algorithm, while for $n >$ *max. depth of the distribution tree* this is identical to the SH algorithm, i.e., no consolidation.)

We evaluate these algorithms in several simulations with random historical messages. The target *location*s are squares, with edge lengths varying from 0.05 to 0.25. The messages contain additional constraints, but due to the nature of the simulated contexts, these actual constraints do not affect the simulation. (We describe the actual client contexts in the following paragraph.) As the ContextNodes in the experiment are uniformly distributed over the simulated area, the size of the target *location* directly affects the number of ContextNodes that may have matching contexts. Figure 6.5 shows the average number of ContextNodes that handle each message over the edge length of the target *location*. As one would expect, the number of ContextNodes is proportional to the area of the target *location*, i.e., the square of the edge length.

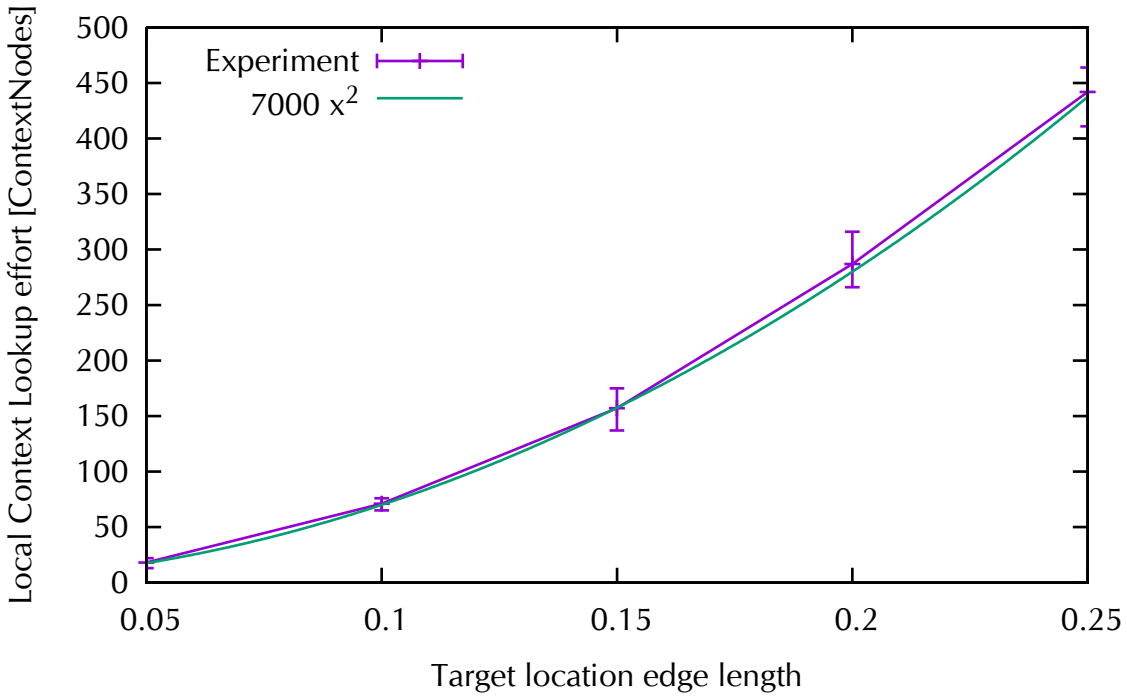


Figure 6.5: Average number of ContextNodes handling Context Lookup for certain target *location* sizes

To reduce the complexity of the simulation, we chose to not simulate individual client contexts and their attributes; many of these contexts that we simulated would never match any message. Thus, their accurate simulation would greatly increase the runtime of each experiment, without contributing to the results. Instead, for each sent historical message, the simulation creates matching contexts as follows: On each of the ContextNodes whose service area intersects with the target *location* for a particular message, between 0 and 300 matching contexts are created. Since we create contexts to match a particular message, they match by definition every constraint of the message. Thus, constraints other than on client *location* do not influence the results of our simulations.

Each of these matching contexts has a different Trusted Node that is responsible for its VID resolution. The selection of TNs again follows a Zipf distribution to account for the fact that usually only a few TNs (20 %) are very popular, i.e., they serve the majority of all clients (80 %).

Note, however, that in each service area is *at most* one matching context per TN, while in reality there are usually several different matching contexts that are handled by the same TN. This simplification does not affect our results, though, since ContextNodes already perform a local duplicate suppression: a message is only forwarded to each TN once, independent of how many contexts that are registered with this TN actually match the message.

Figure 6.6 and Figure 6.7 present our simulation results of these algorithms. As our focus is on the effects of duplicates in the system, we show the message load caused by the different algorithms. The number of messages is shown as the arithmetic mean of ten simulation runs, with errorbars indicating the minimum and maximum number of messages during these runs.

Messages for the Consolidation of Local Results. Figure 6.6 shows the number of messages that result from the consolidation of the local results for the different consolidation levels and for different target area sizes.

Obviously, the FCH algorithm produces the highest message load, which is lower for the algorithms PCH2 through PCH4, since these do not return the local results all the way to the first branching point. Compared to FCH, PCH2 saves between 35 % for target *location*s of 0.05 edge length and 2.93 % for 0.25 edge length. Similarly, PCH3 reduces the number of messages from the collection of results by between 90 % for smaller target *location*s edges of 0.05 and 19.25 % for 0.25 edge length, compared to FCH. Finally, PCH4 lowers the amount of collection messages compared to FCH by between 100 % for 0.05 target *location*s and 58.16 % for 0.25.

Clearly, the larger the target location, the more messages it takes to fully collect and subsequently consolidate the results of the Context Lookup, since more access networks have potentially matching recipients. In these cases, limiting the consolidating depth can significantly lower the amount of associated messages.

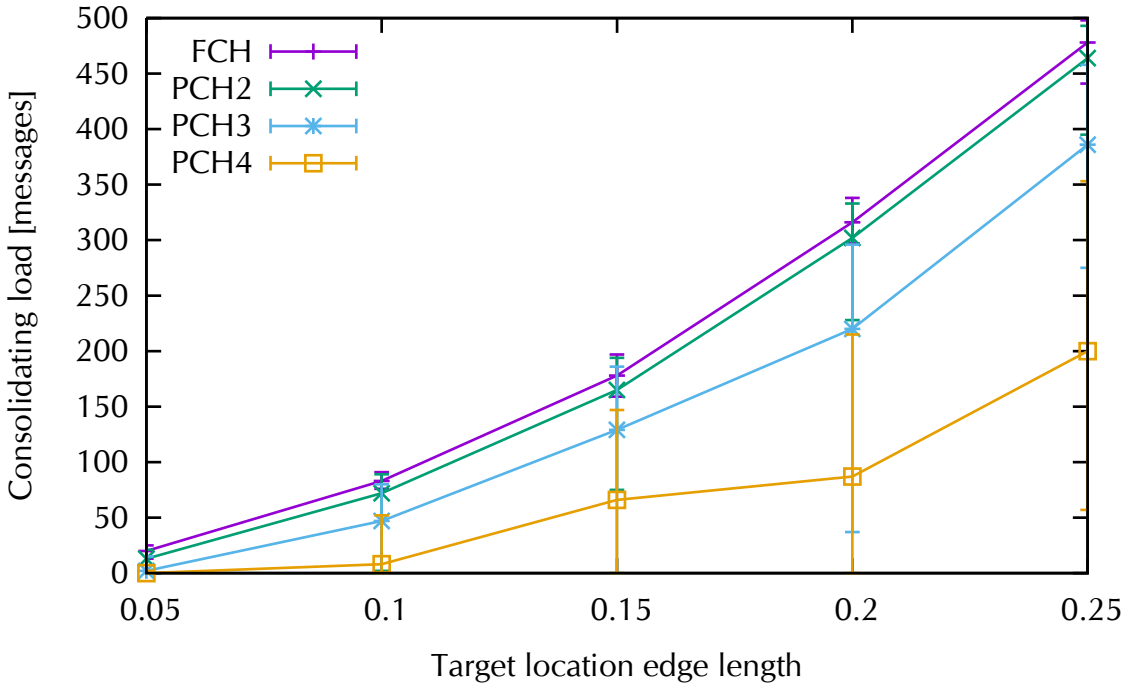


Figure 6.6: Consolidating message load (without VID resolution)

Overall Number of Messages for Historical Routing. Despite the FCH algorithm requiring more messages to collect the results from all access networks, it performs very well due to its ability to eliminate duplicate VIDs and TNs. Figure 6.7 indicates that the total message load (i.e., collecting the results *and* VID Resolution) is lowest for the FCH algorithm. The overall load is between one and two orders of magnitude higher for the SH algorithm, which does not eliminate duplicate VIDs and TNs.

PCH4, which collects local results at the 4th branching point, performs marginally better than SH, at least for *location* sizes of 0.1 and above. This is due to the fact that—for the simulated topology—most distribution trees rarely have more than four branching points, unless the target *location* is huge. Thus, in these trees, almost no collection and consolidation of results occur. Also note that, for very small target *location*s, i.e., 0.05, the results of SH, PCH4, and PCH3 are almost identical. This is due to the fact that with smaller target *location*s, the distribution trees rarely have more than three branching points, which makes PCH4 and PCH3 virtually identical to the SH algorithm. For edge lengths 0.1 and above, PCH3, which collects the results at the 3rd branching point, performs between 50 % and 65 % better than PCH4. Similarly, PCH2 outperforms PCH3 by between 52 % up to 82 %.

These results show that collecting the results is dominated by the effect of VID Resolution and the explicit multi-unicast messages in the overall message load.

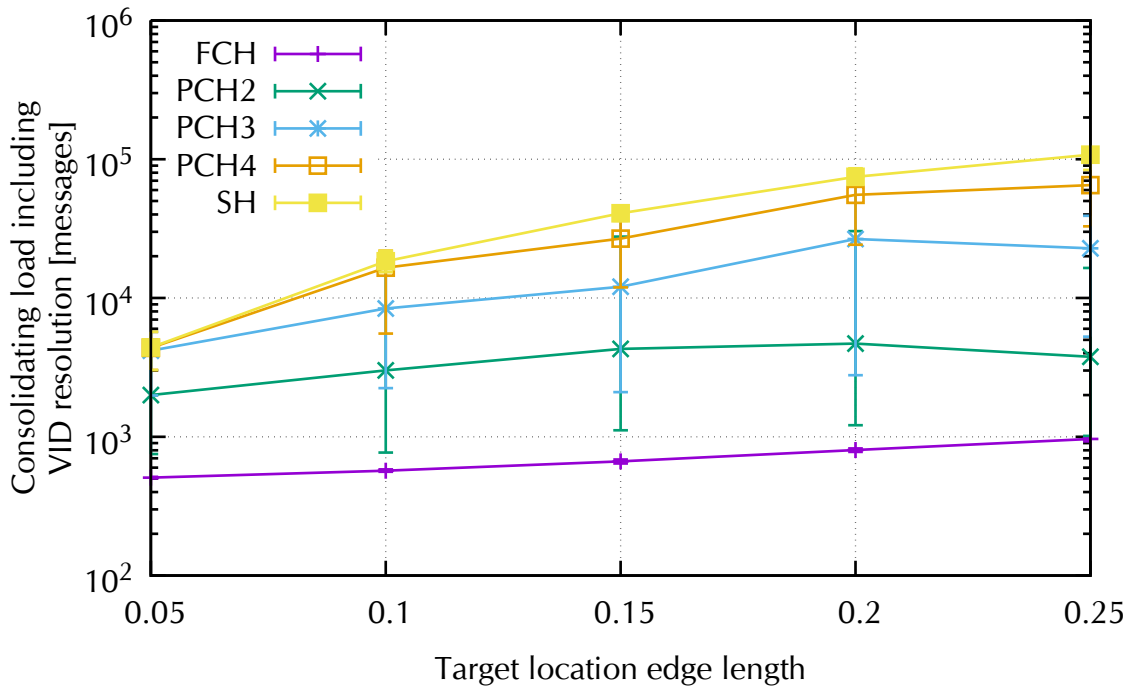


Figure 6.7: Overall historical message load (including consolidating and VID resolution)

Therefore, collecting and consolidating the results of the Context Lookup significantly lowers the message load during the VID Resolution phase of the algorithm. The effect is stronger the more completely the results are collected, i.e., for smaller n in PCH<n> or for FCH. As a result, a Contextcast networks should perform a full consolidation before starting the VID Resolution.

Delay

Obviously, consolidating the results of the Context Lookup adds a delay when routing a historical Contextcast message: First, the results need to be transmitted back to a branching point of the distribution tree. Second, the branching node collects all the TNs from the results. Third, once it has collected the results from all its subtrees, the node can start an explicit multi-unicast to all the TNs with matching recipients.

Our experiments showed that—for the given network topology—FCH in fact increases the average delay for historical messages by 5 overlay hops. We do not expect this delay to be problematic, though. Any increased delay is largely irrelevant since a historical message addresses past contexts, usually several minutes or hours old, perhaps even older. Any additional delay—even if it were in fact several seconds instead of only 5 overlay hops—is not going to affect the delivery in any significant way: Such a delay is relatively small compared to the age of most addressed historical contexts. Either a client is still connected when the message is finally delivered to the resolved entity, then delivery can happen directly to the client's device. Or the client is disconnected from the system, in which case the message is delivered to the client's mailbox. The probability that a client actually disconnects during an additional delay introduced by 5 overlay hops, thus forcing the delivery via the mailbox instead of a direct delivery, is negligible.

Messages Size

Another potentially problematic aspect is the size of the historical messages in the FCH or any of the PCH<n> algorithms. When collecting the results from all access networks with matching VIDs, the header of the explicit multi-unicast message contains the TNs for all matching VIDs. Therefore, the size of a header of such messages increases with the number of matching recipients or rather the number of distinct TNs for these VIDs. However, the number of addressed TNs is limited by the total number of TNs operated in the system, which we expect to be several hundred in reality. Thus, the overhead of the message addressing is still small compared to the size of the message data.

Additionally, the header also includes the set of all VIDs whose context matched the message for the VID resolution. Otherwise, the TNs would need to maintain their own context store for all their clients to determine all matching contexts. First, this duplicates the functionality of the Context Lookup, which already takes place at the ContextNodes (or on another node in the access network). Second, and more importantly, another copy of the client context at its TN compromises the privacy that VIDs provide in the first place. When storing a context under a pseudonym such as the VID in the access network, the ContextNode has no access to the information of what actual client it belongs to. This maintains the anonymity of the client represented by the VID and its associated context. Regular changes of the VID ensure that even cooperating nodes can not easily create a context profile of a client. This changes, however, if a TN has access to the contexts of its clients. The TN can then easily map the contexts to the actual entities, thus profiling its clients. While TNs are trusted to a certain extent, their knowledge needs to be limited to what is required for their particular task, i.e., mapping VIDs to actual clients. This eliminates a source of potential abuse in the system.

While this set of matching VIDs is necessary, it also adds to the size of the explicit multi-unicast header. The number of VIDs is limited by the amount of clients with a matching context, though, and how often a client actually changes its VID. Also, it corresponds directly to the selectivity of the message constraints such as the size of the target *location* or the *time* constraint. If the amount of VIDs becomes problematic for the size of a historical message, a Contextcast system can always impose reasonable limits on these aspects. E.g., a temporal message may only address a certain maximum area, its temporal constraint may not specify more than a certain period of time, or clients may not change VIDs at a rate above a given limit. This way, a temporal Contextcast message matches fewer VIDs, which in turn helps to maintain a manageable header size for the explicit multi-unicast.

Storage Space for Context Archival

In the approach shown, the ContextNodes need to archive the historical contexts of their clients locally. The actual storage requirements depend on the number of clients and the update rate of contexts. We assume that a co-located database is used for the purpose of context storage. Such a database is easily capable of 50,000 inserts of client contexts (i.e., newly registered or changed client contexts) per day; most commercial database systems should be able to handle many more. Such a number of inserts should be sufficient even for popular locations with many transient visitors. With an average size of 2 KB per insert, which is a rather large estimate, this amounts to 100 MB of context data every day. This is well within reach

of today's storage and database technology. Thus, the system should easily scale to, e.g., 10 000 clients with an average of 5 updates per day in a single access network.

If desired, an operator of a Contextcast system can also limit the availability of historical context information to a certain point in the past, e.g., only the last three years. All older contexts could then be removed from the local database, thus limiting the amount of stored data. With the estimate above, this brings the storage requirement down to $3 \times 365\,\text{days} \times 100\,\text{MB/day} = 109\,500\,\text{MB} \approx 106.93\,\text{GB}$. Even with replicated storage, this is well possible with today's storage technology. Considering these numbers, it is even possible to maintain part of this information in main memory for faster access; even though this may not be necessary since delay is of relatively little concern when addressing past client contexts.

In addition to the storage for past client contexts, the Trusted Nodes must also maintain the mapping of VIDs to actual clients. Because of the simplicity of the stored information, the space requirements for TNs is negligible, though. Additionally, if the system limits the storage of old client contexts, this also places a limit on the amount of storage required for storing old VID mappings. Beyond that, if this proves too much data still, the TNs can limit the rate of new VID registrations. E.g., they could offer a basic service with only a certain number of VID registrations during a given time and offer services with higher rates for those customers with increased privacy requirements.

6.4.2 Future Messages

The reactive forwarding of future messages we presented in Section 6.3.3 reduces network load when no client in an access network ever registers a matching context. At the same time, it requires that nodes in the network store copies of messages for potential future recipients. In addition, because messages are only forwarded after a matching recipient is registered, it also increases the message delay. Thus, in the following sections, we provide an analysis of these questions:

1. How does the reactive forwarding affect network load?
2. What are the storage requirements for the reactive forwarding of future messages?
3. What is the effect of reactive forwarding on the delay between registering a context and receiving a future message?

Message Load

Obviously, a reactive forwarding minimizes the amount of false positives. The actual number of saved messages depends on (1) the actual addressing of messages, (2) the

client contexts in the system, and (3) the network topology, i.e., how many false positives a system would have forwarded without it.

Let M be a future Contextcast message; let $C_{\text{cand}} = \{C_1, \ldots, C_k\}$ be the set of all contexts that are registered in a single access network inside of M's target location in the time frame that matches M's temporal constraint; and finally, let A denote the event $\forall C \in C_{\text{cand}} : C \not\sqsubset M$.

Thus, let $P(A) = p_f$ denote the probability of A, i.e., that none of a set of candidate contexts matched the contexts matched the constraints of M. Consequently, $P(\bar{A}) = 1 - p_f$ is the probability that there is at least one matching recipient in the candidate set C_{cand}. This in turn requires that M is forwarded to this access network.

In a system with contexts that have uniformly distributed *location*s, we can deduce from this that a given message needs to be forwarded only to a fraction $1 - p_f$ of the access networks intersecting the addressed area to be delivered to a client. Compared to the simpler approach, which forwards the message to every ContextNode whose service area intersects the target *location*, storing messages close to the source ContextNode avoids forwarding the message to p_f of the access networks.

How much this actually reduces network load depends on the network topology and the resulting distribution tree. In the worst case this could nullify any saving from the reactive forwarding (e.g., if the topology were just a linear sequence of routers), in reality we expect the topology to be rather tree-like and thus more favorable for our approach.

Also, as we argued in Chapter 4, we expect a certain clustering of similar clients at geographic locations. Thus, once a context matching a certain message is registered at a ContextNode and the message forwarded to this ContextNode, it is then already stored locally for all matching contexts that register at the same ContextNode at a later time. Additionally, it may also be stored at close branching points, thus reducing the necessary message load if a matching context registers at a nearby ContextNode.

Storage Space on ContextRouters

The reactive forwarding of future messages increases the storage requirement for individual nodes on the path between sender and ContextNodes in the target *location*.

However, the approach does not lead to higher storage requirements for the system as a whole; in fact, the overall storage requirements are the same or lower: An inner node in the distribution tree stores a message if and only if (1) there is at least one ContextNode with potentially matching future contexts within one of its subtrees, and (2) it has not forwarded the message towards this ContextNode, yet.

Thus, for each ContextNode in a distribution tree that stores a copy for the reactive forwarding at least one ContextNode does not need to store this message. The approach saves more storage space if it stores a message at an inner node for multiple subtrees (and therefore leaves) with potentially matching future contexts.

Again, the approach particularly benefits from the locality in our Contextcast overlay network coupled with local similarity of clients: When two or more ContextNodes' service areas intersect the target *location* of a future Contextcast message, a message is forwarded to an access network once a matching context is registered there. At the same time, the message is stored at a nearby node along the path matching clients in neighboring service areas.

In absolute numbers, the storage requirements for the reactive routing depends only on the messages. As we discussed in Chapter 2, the Contextcast network of overlay routers consists of server-type computer hardware until appropriate support may be provided by common IP router vendors. A ContextRouter with current technology therefore may well have access to 100 GB of persistent storage space. With average message sizes of 100 KB, each router can store 1 000 000 messages at any time. This is a conservative estimate, as most payloads will likely be less. Additionally, messages can always reference additional content to be loaded on demand for particularly bandwidth-intensive applications; this also benefits clients that are only interested in some messages of a particular type, not all of them.

The majority of temporal predicates limits the lifetime of messages, as there is a definitive point after which no context can match the temporal constraint (see Table 6.2). However, depending on the actual constraint, this can still be arbitrarily far in the future. An operator of a Contextcast system can and should additionally impose a maximum message lifetime to limit the required storage capacity for future messages. If the message lifetime is limited to a maximum of 60 days, the storage capacity and message size estimates equate to more than 5500 new future messages per day that each ContextRouter can store. This maximum lifetime is rather long, considering that the content of most messages is outdated long before 60 days have passed.

If a ContextRouter runs out of storage space at any time, it can forward messages towards the leaves in the distribution tree. Routers further down then become responsible to store the messages for future recipients. This process can be repeated until a message reaches a router that can still store new messages. If there is none, it eventually reaches an access network, where additional storage space is available because of the need to store historical user contexts.

If a ContextRouter forwards messages for this reason without a known recipient, it should prefer to forward newer messages. The rationale is that if an older message didn't find a recipient until such a time, the probability is higher that it will not find one in its remaining lifetime. Thus forwarding it at this time would likely lead to

an increase in false positive load. Obviously, one can imagine other, more complex ways to select messages in such a case, such as statistics on context properties to estimate the probability of a given message finding a matching recipient in the future. However, this is a topic for future research.

Message Delay

Compared to a proactive approach that disseminates future messages ahead of time, our reactive approach suffers an increased message delay for the first matching recipient in an access network. This delay depends on the path between the message source and the recipient. At most, it is the round-trip time between message source and recipient: First, the newly registered context is forwarded throughout the network until it reaches a ContextRouter where the message is stored for future recipients. Second, the message is then forwarded to the matching recipient; this step takes advantage of the tag with matching VIDs to speed up the forwarding towards already matched recipients. Of course, this may be shorter if the message has been partially disseminated before and a copy is stored on one of the intermediate ContextRouters.

In general, we consider the reactive forwarding delay negligible since (1) it only occurs for the first matching recipient in an access network, and (2) it is typically much smaller than the delay between sending a future message and the registration of a matching recipient. In theory, it might happen that a recipient disconnects again before a reactively forwarded future message is delivered. In practice, such extremely short connection spans are highly unlikely, though. They would affect the system negatively in many other areas as well.

6.5 Related Work

The systems that we discussed in Section 2.3 largely neglect the issue of time. Most of the systems provide only implicit temporal support in their semantics.

Multicast approaches, both infrastructure-based, such as [BFC93, DEF+96], and application layer approaches, such as [CRSZ02, Cha03], provide no support for temporal relations. Messages are sent to those recipients that had joined a group at the time a message is sent to this group. It is, however, not possible to restrict messages to a certain time period, e.g., to all clients that had already joined 5 min ago. Additionally, it is usually required that the join message has been propagated to the sender of a message. This is due to the same reasons we discussed in section 3.3.2 for a distributed Contextcast system: Intermediate routers require this information to forward messages accordingly towards the members of a group.

Similarly, geographical addressing and routing approaches, such as [Nav01, Rot03, Dür10, Heu05, KLS07], rely only on the time a message is sent for its delivery: a message is delivered to those recipients that are within the target area at the time the message reaches an access network. Due to the delay when forwarding a message, this is usually some time after the message was sent. It is, however, not possible to restrict recipients temporally, i.e., only those that are in the target area during a certain time period. However, due to the nodes' assignment of service areas, these systems do not require a client's information to be propagated before a message can be disseminated.

The same applies to Pub/sub systems in general. Almost all such systems are based on an implicit "future" semantics. Subscriptions, either topic-based such as [TIB08], type-based such as [EGD01, PB02, Pie04], or content-based such as [SA97, Car98, CRW00], implicitly subscribe to events that happen after the subscription is issued, i.e., in the future. (As before, delivery in a distributed Pub/sub system also requires that subscriptions are propagated to the sources before brokers can forward matching events.) Almost no Pub/sub system allows temporal restrictions in subscriptions, other than the implicit "future" semantics mentioned before.

PADRES, introduced in [FJLM05], is a Pub/sub system, though, that supports clients to also retrieve notifications of past events [LCH+07]. The authors achieve this by storing past events in databases and allowing subscriptions to specify temporal predicates. Similarly to what we present in this chapter, this allows clients to subscribe to historical events, future events, and hybrid events, which are a combination of historical and future events. In their approach, the databases themselves are merely subscribers, issuing subscriptions for the events they want to archive. This allows the system to store a subset of past events on databases and partition the event space among databases flexibly. The broker system handles subscriptions, either with or without temporal constraints, in the usual fashion, routing them towards the senders established by advertisements. This way, brokers can update their notification forwarding tables with the subscription or, in the case of a subscription to historical events, one or more databases with matching events can return the results of the query.

This design is similar to our approach in Section 6.3.2. However, due to the differences between Contextcast and Pub/sub, Contextcast needs to store historical contexts, not messages, which are the equivalent of events in Pub/sub. This, together with the need to deliver messages to historical clients, poses a privacy challenge that is not present in a Pub/sub system. In PADRES, the stored events contain no information that clients could not have obtained by simply subscribing to and storing them themselves.

For scalability and simplicity, the temporal Contextcast approach we presented

archives historical contexts spatially partitioned directly on the ContextNodes where a client had registered its context, i.e., where the contexts had originated in the system. When disseminating historical messages, a spatial index is used to forward a message to the service areas with potentially matching contexts. It is possible to extend Contextcast with distributed indexes similar to a Distributed Hash Table such as Chord [SMLN+01], Pastry [RD01], Tapestry [ZKJ01], or CAN [RFH+01]. Such an index can then be used if a message contains an overly large *location* constraint or none at all. The routing performance of a DHT-based index is expected to be comparable to that of the spatial index, provided the mapping of attribute values to nodes preserves the locality of the attribute values. The maintenance of such indexes causes additional overhead, though, to place the item on the corresponding node for the chosen partitioning of the values. One can therefore even imagine an adaptive approach where the system compares the spatial index and alternative attribute indexes and establishes additional indexes if the spatial index performs poorly due to the prevalent *location* constraints in the system.

While most Pub/sub systems restrict subscriptions to single events, especially in large scale systems clients may be interested in the occurrence of a set of events. An extension to this paradigm is Complex Event Processing (CEP), with *complex* or *composite events* that are published whenever a certain set of events is detected [PSB03, PSB04]. A specialized composite event language provides the means to specify the set of interesting events and their relation. For instance, an application may be interested in a composite event generated by a sequence of events A and B, i.e., A occurs before B. Or a timing, i.e., event B (not) occurring within a period of time after event A. However, due to the different goal, such composite event detection provides only limited temporal support, suitable for their particular purpose.

The authors of [DFST11] present a system that can efficiently correlate live event streams with archived ones. To this end, instead of computing all possible patterns, the system computes all possible patterns on one side (e.g., the live stream) but only the necessary pattern on the other side. However, the focus lies again on the correlation of events with certain relations, not temporal aspects of subscriptions or event dissemination.

The support for addressing historical and future contexts in Contextcast requires a representation of time and temporal relations to specify matching clients temporally. One of the most well-known and influential approaches in this area is the work of [All83]; it goes beyond classical approaches in database management systems, which index facts by their date, i.e., a representation of time that allows for a temporal ordering of facts with simple operations such as comparisons on integer timestamps [Bru72, Hen74], or systems based on before/after chains such as [Bru72], which allow for the easy expression of relative, imprecise temporal relations but quickly reach their limits as the amount of temporal information grows. To overcome

these difficulties, Allen's system works with intervals as primitives and a system of 13 relations to express temporal relationships, on which he builds an inference approach for temporal knowledge.

Based on these relations, we have developed a set of temporal predicates and incorporated in our system (see Section 6.3.1 for details). They allow Contextcast to express imprecise temporal relations in a very natural way, thus making these predicates easy to use for human users of our system.

6.6 Summary

In this chapter, we showed how to extend Contextcast with support for temporal concepts. This extends the original paradigm in two useful ways beyond the original semantics we presented in Chapter 3: First, it allows to send historic Contextcast messages to entities that had a certain context in the past, without the need for explicit recipient addresses or client subscriptions. Second, it enables senders to address Contextcast messages to entities that register a given context some time in the future. In addition to the semantic changes, we have shown algorithms to efficiently disseminate both historic and future messages on the basis of our Contextcast overlay network.

The delivery of historic messages requires a history of client contexts. We use a distributed storage of historic context information in the respective access networks; the service area coverage serves as a geographical index over these stored contexts. To eliminate the possibility of client profiling, the stored contexts use Virtual Identities (VIDs) to identify the corresponding client. A number of trusted third parties, the Trusted Nodes (TNs), are responsible for creating the VIDs and for resolving the corresponding clients during message delivery. The historic routing algorithm avoids duplicate VIDs and sends only a single message to each of the TNs. To accomplish this, the FCH algorithm collects all results from the Local Context Lookup phase, removes duplicate VIDs and addresses the message to each of the TNs that has matching contexts registered.

Future messages can delay message forwarding until a matching context is registered in the network. To this end, ContextRouters determine the downstream routers in the dissemination tree according to the target location. They can then store messages for as long as there is no matching recipient known from one of its subtrees. This ensures that messages are not forwarded when no matching context is ever registered.

In the evaluation of these concepts we found that the collection of local results in the FCH algorithm reduces the overall network load by between 88 % and 99 % compared to the SH algorithm; compared to the PCH<n> algorithms, which do

not fully collect the local results, it can still save between 74 % and 98 % of overall network load. This is due to the FCH algorithm's ability to eliminate both duplicate VIDs and TNs before the VID Resolution phase. The storing of future messages at nodes close to the sender minimizes the amount of unnecessarily forwarded messages if no matching context is ever registered. It causes, however, a slightly increased delay when the first matching clients registers in an access network. The approach does not require additional storage in the overlay network, though: for every ContextRouter that stores a message, there is at least one access network where potential matching recipients may register in the future, which does not need to store the message for these potential recipients at that time.

Chapter 7

Conclusion

Ceterum censeo Carthaginem esse delendam

(Cato Censorius)

In this chapter, we provide an overview of the results of this thesis and an outlook on possible research directions in this field.

7.1 Summary

The continuously increasing power of mobile devices along with more and more sensors to capture a client's context enables an ever increasing number of context-aware applications and services. One such service, which can be used as a foundation for other context-aware services and applications, is a context-aware communication mechanism or Contextcast. It facilitates senders to disseminate messages to clients with a specified context.

We presented and classified related approaches to efficiently disseminate messages to a set of recipients and showed their respective shortcomings for their use in contextual message dissemination. Based on these results, we developed the Contextcast system and its individual parts: First, client contexts, which represent a client's current context in terms of a set of context attributes. Second, messages, which specify the set of recipients using constraints on context attributes and also contain a payload with application specific data. Third, the semantics that matches client contexts to the addressing of contextual messages. And fourth, a reference dissemination algorithm, which uses client context information for a directed dissemination of messages towards recipients that match the addressing.

One drawback of the reference dissemination algorithm, though, is its requirement for complete and up-to-date context information on all the ContextRouters. In a global system with many clients, this requirement causes the propagation of a

large number of context updates, thus severely limiting the system's scalability. To increase the system scalability while preserving Contextcast's semantics, we developed two improved routing mechanisms which reduce the amount of context information and context updates.

The first advanced routing scheme uses coarser context information for routing. For the clients' *location*, such a coarser representation is readily available via the ContextNode's service area coverage. It is more challenging to find a coarse representation for other context attributes, though. To overcome this challenge, we proposed a scheme to derive a coarse representation automatically from the similarity of client contexts. To this end, ContextRouters can collect similar contexts into a single, aggregated one. Obviously, the effectiveness of routing with aggregated context information depends on the similarity of currently registered clients. If there are many similar contexts, they can easily be replaced by a single aggregated context. However, if all contexts are rather distinct, more contexts need to be propagated to the routers separately, thus lowering the benefit over the reference algorithm from Chapter 3. In the extreme, all pairwise contexts are sufficiently different so no aggregation occurs. In this case, the aggregated routing scheme is practically identical to the reference algorithm, which propagates each context individually.

The trade-off when using aggregated context information for directed forwarding, however, is that ContextRouters need to forward a message if there is an aggregation that contains a potentially matching context; some of these messages might not reach a matching recipient in the end, thus becoming a false positive. Despite this, using aggregated context information lowers the overall system load, i.e., updates and messages, by between 18 % and 25 %, in some scenarios up to almost 30 %, compared to the reference dissemination; this improvement was achieved even though there was no pronounced similarity of client contexts.

The second advanced approach is orthogonal to using aggregated context information for Contextcast routing; it adaptively propagates client contexts only towards message sources whose messages may match that particular context. To this end, nodes record statistics about the observed messages and contexts. From this information, they can derive whether the propagation of a given piece of context information is beneficial to the overall system load: The routers compare the load it takes to keep a piece of context information up-to-date to the load that is generated when a particular class of messages is forwarded speculatively without context knowledge, simply on the assumption that a matching recipient exists in a direction.

Similarly to the directed forwarding with aggregated information, this speculative forwarding ensures that the Contextcast semantics remains intact. But in the same way, these speculatively forwarded message may not reach a matching recipient eventually, thus increasing the overall system load again after lowering the amount of updates. In our experiments, the adaptive propagation of contexts lowers the

overall system load by more than 40 %; this is largely the result of a strong decrease in update load with a moderate increase in message load due to false positives.

In addition to the basic semantics we presented in Chapter 3, we also extended Contextcast to allow for a temporal addressing of recipients. This allows us to overcome the implicit "right now" semantics, which would disseminate a message only to those clients that are connected to the system at the time the message is sent and whose context had already been propagated to the message source. A newly introduced *time* attribute allows senders to add a temporal constraint to messages, which recipients must also match for a message to be delivered to them. A set of corresponding temporal predicates provides clients and application developers with a very intuitive way of specifying the temporal constraint.

Besides the required semantic modification, we also presented optimized routing algorithms for such temporal messages. historical message routing, i.e., messages addressing historical contexts, focuses on the privacy of clients and an efficient lookup of recipients despite the overhead necessary to protect the clients' privacy. To this end, routers collect the matching clients (or rather, their Virtual Identities (VIDs)) and eliminate duplicate VIDs and Trusted Nodes. Our experiments showed that this is able to reduce the network load by between 88 % and 99 %, compared to a simple version that does not remove such duplicates. Future message routing, in contrast, does not require any particular privacy consideration since messages can be delivered directly once a client connects in the future and their context matches a message. However, to prevent the system from forwarding messages that no context ever matches in the future, we employ a reactive forwarding system, where routers store copies of the message for potentially matching recipients later on. We analyzed this approach and showed that it does not increase the required storage capacity and merely moves the stored messages from the ContextNodes in the target *location* to nodes in the overlay. This obviously reduces the network load if no matching context registers in an access network during a messages *time* constraint.

In conclusion, the presented Contextcast semantics and the corresponding routing algorithms provide the basis of a contextual communication mechanism. It enables an efficient and flexible message dissemination scheme, which can be incorporated in large-scale context-aware systems such as Nexus [GBH+05, REF+06, LCG+09].

7.2 Outlook

There are several possible paths to extend the work presented in this thesis in future research.

Self-tuning Routing Algorithms

The presented routing algorithms based on coarse context information and adaptive propagation of information are already designed to adapt to the observed contexts and messages: First, our aggregation of client contexts derives the coarser, aggregated contexts from the observed singleton contexts. Second, the adaptive propagation of context information relies on statistics of message addressing and context updates to determine what context information benefits overall system load in a directed forwarding.

In both approaches, an administrator can influence the algorithm by adjusting parameters. The similarity threshold S_{th} influences how coarse or fine the routers aggregate contexts. Similarly, the propagation and invalidation threshold, $B_{\text{th,P}}$ and $B_{\text{th,I}}$, respectively, determine by how much a composite composite must lower or increase the system load before it is propagated or invalidated, respectively.

An automated adjustment of these parameters would allow the system to autonomously react to changes in message and context update rates: With low message and high update rates, system load is dominated by updates. Message load, in contrast, is low, thus also false positives have little effect. In this case, lowering the similarity threshold decreases the update load by propagating coarser contexts; similarly, a higher propagation threshold causes routers to propagate fewer composite contexts, which need to be kept up-to-date. At the same time, these changes increase the amount of false positives slightly, due to the coarser context information or increased speculative forwarding. This is of little consequence for low message rates, though. In the opposite case, for high message and low update rates, a higher similarity threshold or a lower propagation threshold provides routers with better information for a directed forwarding. This reduces the amount of false positives and speculative forwarding. The required update load due to the finer and further propagated context information is largely irrelevant for low update rates.

(Network) Distance-aware Context Information

In its current form, the proposed context aggregation is unaware of the distance between message sources and clients. Thus, client contexts are propagated throughout the network, independent of the distance from the node where they are registered. In practice, this causes a high update load far from a context's origin, with little benefit to the directed forwarding. A message originating on another continent, for instance, could be forwarded speculatively until it is closer to the target access networks. The dissemination tree in such a case usually starts rather linear and branches close to the destination, i.e., somewhere on the destination continent. The load caused by speculatively forwarding a message closer to the target nodes is

small compared to the load caused by keeping a large number of client contexts up-to-date throughout the network.

We can exploit this in a couple of ways. First, the network distance of contexts from their source can be incorporated in the aggregation process. Thus, contexts that have been propagated farther into the network get aggregated more aggressively, leading to coarser context information farther away from the context origin. Second, the adaptive propagation of client contexts already incorporates a similar idea implicitly. Routers far from the actual destination access networks hardly ever regard a message as false positive, as long as there is only a single matching recipient for it in the end. Thus, they establish fewer composite contexts for a directed forwarding farther from the destination.

The routing could be improved further, however, by introducing a hierarchical network design for Contextcast. Then, routers can limit the precise propagation of contexts to smaller domains (towns, states, or maybe countries). On a larger scale (countries and continents), they can maintain a few indexes of highly selective or often used attributes, such as *location*. These are then used to forward a message closer to far away destination, where more precise routing information is available.

Negative Comparisons with Partial Information

When using incomplete information for forwarding, routers must assume a matching recipient to avoid false negatives. To overcome this limitation, we introduced the concept of composite contexts, which contain complete knowledge for a given set of attributes. This complete knowledge can then be used to deduce the absence of a matching value for some attributes.

A different approach would be to allow a new kind of routing information, which provides nodes with the knowledge that a particular value or combination does not exist somewhere. A prime example for this is the *location* attribute: if routers have the information of what locations can be reached over a link, they can test the *location* constraint, directly pruning all branches which have no overlap with the destination *location*. The concept can be applied to any highly selective attribute, though. If a router continuously receives messages addressed to people of *age* 61, for which it has no knowledge of matching recipients, it might propagate the information that "there are no recipients of *age* 61 reachable via this link". Keeping such a piece of information up-to-date causes less load than propagating complete knowledge of all contexts that contain the *age* attribute.

Aggregation of Historical Contexts

In its proposed form, the delivery of historical Contextcast messages requires a complete history of client contexts. While Virtual Identities (VIDs) protect clients' privacy, the complete history raises the complexity of the management and the local lookup. We have already proposed to limit the availability of historical context information for storage space reasons.

An alternative solution would be the adaptation of the aggregation concept, which was developed for current routing information, to historical context information. This would allow ContextNodes to aggregate older context information to lower the amount of stored data and improve the lookup of matching contexts. Obviously, this introduces a certain amount of false positives due to the uncertainty that results from the aggregation. This approach, however, allows historical Contextcast messages without imposing a limit for the age of contexts that can be addressed. And since messages that address very old contexts are usually rather rare, the additional false positive are mostly negligible.

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AO	Active Object
ALM	Application Layer Multicast / Application Level Multicast
API	Application Programming Interface
AWM	Augmented World Model
CAN	Content Addressable Network
CEP	Complex Event Processing
CP	Context Provider
DB	Distance-based (Reporting)
DBR	Distance-based Reporting
DFG	Deutsche Forschungsgemeinschaft
DHT	Distributed Hash Table
DNS	Domain Name System
DR	Dead Reckoning
DVMRP	Distance Vector Multicast Routing Protocol
FCH	Fully Consolidating Historical
FN	Federation Node
FQDN	Fully Qualified Domain Name
GiST	Generalized Search Tree
GPS	Global Positioning System
HSDPA	High-Speed Downlink Packet Access

ID	identifier
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISP	Internet Service Provider
JEDI	Java Event-based Distributed Infrastructure
LAN	Local Area Network
LBS	Location-based Service
LTE	Long Term Evolution
LSI	Location Server Infrastructure
MOD	Moving Objects Database
MWS	Middleware Service
NPO	Nonprofit Organization
NTP	Network Time Protocol
OSPF	Open Shortest Path First
P2P	Peer-to-Peer
PCA	Pairwise Context Aggregation
PCH	Partially Consolidating Historical
PIM-SM	Protocol Independent Multicast Sparse Mode
PST	Parallel Search Tree
Pub/sub	Publish/subscribe
RIP	Routing Information Protocol
SDC	Source Description Class

SAA	Service Area Approximation
SFB	Sonderforschungsbereich
SFF	Siena Fast Forwarding
SH	Simple Historical
SMA	Simple Moving Average
SPCF	Shortest Path Context Forwarding
TN	Trusted Node
UMTS	Universal Mobile Telecommunications System
UTC	Coordinated Universal Time
VID	Virtual Identity
WGS84	World Geodetic System 84
WLAN	Wireless Local Area Network

Bibliography

[AEM99] Marcel Altherr, Martin Erzberger, and Silvano Maffeis, *iBus - A Software Bus Middleware for the Java Platform*, Proceedings of the Workshop on Reliable Middleware Systems of IEEE SRDS'99, 1999, pp. 43–53.

[AHWY03] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu, *A Framework for Clustering Evolving Data Streams*, Proceedings of the 29th international conference on Very large data bases, VLDB'03, VLDB Endowment, 2003, pp. 81–92.

[All83] James F. Allen, *Maintaining Knowledge about Temporal Intervals*, Communications of the ACM **26** (1983), no. 11, 832–843.

[ASS+99] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra, *Matching Events in a Content-based Subscription System*, Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, PODC '99, ACM, 1999, pp. 53–61.

[BBK02] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy, *Scalable Application Layer Multicast*, Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '02, ACM, 2002, pp. 205–217.

[BBMS98] John Bates, Jean Bacon, Ken Moody, and Mark Spiteri, *Using Events for the Scalable Federation of Heterogeneous Components*, Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications, EW 8, ACM, 1998, pp. 58–65.

[BCM+99] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman, *An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems*, Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, ICDCS'99, IEEE Computer Society, 1999, pp. 262–272.

[BCQ+07] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca, *A Data-oriented Survey of Context Models*, ACM SIGMOD Record **36** (2007), no. 4, 19–26.

[BDR07] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg, *A survey on context-aware systems*, International Journal of Ad Hoc and Ubiquitous Computing **2** (2007), no. 4, 263–277.

[BFC93] Tony Ballardie, Paul Francis, and Jon Crowcroft, *Core Based Trees (CBT)*, ACM SIGCOMM Computer Communication Review **23** (1993), no. 4, 85–95.

[BFI+07] Rick Boivie, Nancy Feldman, Yuji Imai, Wim Livens, and Dirk Ooms, *Explicit Multicast (Xcast) Concepts and Options*, RFC 5058 (Experimental), November 2007.

[BFM00] Rick Boivie, Nancy Feldman, and Christopher Metz, *Small Group Multicast: A New Solution for Multicasting on the Internet*, IEEE Internet Computing **4** (2000), no. 3, 75–79.

[BH67] Geoffrey H. Ball and David J. Hall, *A clustering technique for summarizing multivariate data*, Behaviorial Sciences **12** (1967), no. 2, 153–155.

[Blo70] Burton H. Bloom, *Space/Time Trade-offs in Hash Coding with Allowable Errors*, Communications of the ACM **13** (1970), no. 7, 422–426.

[Bru72] Bertram C. Bruce, *A Model for Temporal References and Its Application in a Question Answering Program*, Artificial Intelligence **3** (1972), 1–25.

[Car98] Antonio Carzaniga, *Architectures for an Event Notification Service Scalable to Wide-area Networks*, Ph.D. thesis, Politecnico di Milano, December 1998.

[CD85] David R. Cheriton and Stephen E. Deering, *Host Groups: A Multicast Extension for Datagram Internetworks*, ACM SIGCOMM Computer Communication Review **15** (1985), no. 4, 172–179.

[CdC05] Gianpaolo Cugola and Jose Enrique Munoz de Cote, *On Introducing Location Awareness in Publish-Subscribe Middleware*, 25th IEEE International Conference on Distributed Computing Systems Workshops, ICDCSW'05, IEEE Computer Society, 2005, pp. 377–382.

[Cha03] Yatin Chawathe, *Scattercast: an adaptable broadcast distribution framework*, Multimedia Systems **9** (2003), no. 1, 104–118.

[ČJP05] Alminas Čivilis, Christian S. Jensen, and Stardas Pakalnis, *Techniques for Efficient Road-Network-Based Tracking of Moving Objects*, IEEE Transactions on Knowledge and Data Engineering **17** (2005), no. 5, 698–712.

[CM08] Gianpaolo Cugola and Matteo Migliavacca, *On Context-Aware Publish-Subscribe*, Proceedings of the International Conference on Distributed Event-based Systems, DEBS '08, 2008, (Fast abstract), pp. 1–2.

[CMM09] Gianpaolo Cugola, Alessandro Margara, and Matteo Migliavacca, *Context-Aware Publish-Subscribe: Model, Implementation, and Evaluation*, IEEE Symposium on Computers and Communications, ISCC 2009, IEEE, July 2009, pp. 875–881.

[CNF01] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta, *The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS*, IEEE Transactions on Software Engineering **27** (2001), no. 9, 827–850.

[CRSZ02] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang, *A Case for End System Multicast*, IEEE Journal on Selected Areas in Communications **20** (2002), no. 8, 1456–1471.

[CRW99] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf, *Interfaces and Algorithms for a Wide-Area Event Notification Service*, Technical Report CU-CS-888-99, Department of Computer Science, University of Colorado, October 1999, revised May 2000.

[CRW00] ———, *Content-Based Addressing and Routing: A General Model and its Application*, Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, Boulder, CO 80309, USA, January 2000.

[CRW01] ———, *Design and Evaluation of a Wide-Area Event Notification Service*, ACM Transactions on Computer Systems (TOCS) **19** (2001), no. 3, 332–383.

[CW03] Antonio Carzaniga and Alexander L. Wolf, *Forwarding in a Content-Based Network*, Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '03, ACM, 2003, pp. 163–174.

[DA99] Anind K. Dey and Gregory D. Abowd, *Towards a Better Understanding of Context and Context-Awareness*, Technical Report GIT-GVU-99-22, Graphics, Visualization and Usability Center and College of Computing, Georgia Institute of Technology, 1999.

[DBR05] Frank Dürr, Christian Becker, and Kurt Rothermel, *Efficient Forwarding of Symbolically Addressed Geocast Messages*, Proceedings of the 14th International Conference on Computer Communications and Networks, ICCCN'05, IEEE, October 2005, pp. 77–83.

[DBR06] ———, *An Overlay Network for Forwarding Symbolically Addressed Geocast Messages*, Proceedings of the 15th International Conference on Computer Communications and Networks, ICCCN'06, IEEE, October 2006, pp. 427–434.

[DC90] Stephen E. Deering and David R. Cheriton, *Multicast Routing in Datagram Internetworks and Extended LANs*, ACM Transactions on Computer Systems (TOCS) **8** (1990), no. 2, 85–110.

[DEF+96] Stephen Deering, Deborah L. Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei, *The PIM Architecture for Wide-Area Multicast Routing*, IEEE/ACM Transactions on Networking (TON) **4** (1996), no. 2, 153–162.

[DFST11] Nihal Dindar, Peter M. Fischer, Merve Soner, and Nesime Tatbul, *Efficiently Correlating Complex Events over Live and Archived Data Streams*, Proceedings of the 5th ACM International Conference on Distributed Event-based Systems, DEBS '11, ACM, 2011, pp. 243–254.

[DPG+08] Frank Dürr, Jonas Palauro, Lars Geiger, Ralph Lange, and Kurt Rothermel, *Ein kontextbezogener Instant-Messaging-Dienst auf Basis des XMPP-Protokolls*, 5. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste, Sonderdruck Schriftenreihe der Georg-Simon-Ohm-Hochschule Nürnberg, vol. 42, Georg-Simon-Ohm-Hochschule Nürnberg, September 2008, pp. 23–28.

[DR03] Frank Dürr and Kurt Rothermel, *On a Location Model for Fine-Grained Geocast*, Proceedings of the 5th International Conference on Ubiquitous Computing 2003, Lecture Notes in Computer Science, vol. 2864, Springer-Verlag Berlin Heidelberg, 2003, pp. 18–35.

[DR08] ———, *An Adaptive Overlay Network for World-wide Geographic Messaging*, Proceedings of the 22nd IEEE International Conference on Advanced Information Networking and Applications, AINA 2008, IEEE, March 2008, pp. 875–882 (Englisch).

[Dür10] Frank Dürr, *Geographische Kommunikationsmechanismen auf Basis von feingranularen räumlichen Umgebungsmodellen*, Dissertation, Universität Stuttgart, 2010.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, *The Many Faces of Publish/Subscribe*, ACM Computing Surveys (CSUR) **35** (2003), no. 2, 114–131.

[EGD01] Patrick Th. Eugster, Rachid Guerraoui, and Christian Heide Damm, *On Objects and Events*, Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, OOPSLA '01, ACM, 2001, pp. 254–269.

[ESRM03] Ayman El-Sayed, Vincent Roca, and Laurent Mathy, *A Survey of Proposals for an Alternative Group Communication Service*, IEEE Network **17** (2003), no. 1, 46–51.

[FGKZ03] Ludger Fiege, Felix C. Gärtner, Oliver Kasten, and Andreas Zeidler, *Supporting Mobility in Content-Based Publish/Subscribe Middleware*, Middleware 2003, ACM/IFIP/USENIX 2003 International Middleware Conference, Lecture Notes in Computer Science, vol. 2672, Springer-Verlag Berlin Heidelberg New York, June 2003, pp. 103–122.

[Fin87] Gregory G. Finn, *Routing and Addressing Problems in Large Metropolitan-scale Internetworks*, ISI Research Report ISI/RR-87-180, Information Sciences Institute, University of Southern California, March 1987.

[FJL+01] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha, *Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems*, ACM SIGMOD Record **30** (2001), no. 2, 115–126.

[FJLM05] Eli Fidler, Hans-Arno Jacobsen, Guoli Li, and Serge Mankovski, *The PADRES Distributed Publish/Subscribe System*, Proceedings of the 8th International Conference on Feature Interactions in Telecommunications and Software Systems, ICFI'05, IOS Press, 2005, pp. 12–30.

[FKP02] Alex Fabrikant, Elias Koutsoupias, and Christos H. Papadimitriou, *Heuristically Optimized Trade-Offs: A New Paradigm for Power Laws in the Internet*, Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 2380, Springer-Verlag Berlin Heidelberg, 2002, pp. 110–122.

[FLR07] Tobias Farrell, Ralph Lange, and Kurt Rothermel, *Energy-efficient Tracking of Mobile Objects with Early Distance-based Reporting*, Proceedings of the Fourth Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, MobiQuitous 2007, IEEE, August 2007, pp. 1–8.

[Fra00] Paul Francis, *Yoid: Extending the Internet Multicast Architecture*, Unpublished paper, available at http://www.aciri.org/yoid/docs/index.html [online 2015-11-04], ACIRI, 2000.

[GBH+05] Matthias Grossmann, Martin Bauer, Nicola Hönle, Uwe-Philipp Käppeler, Daniela Nicklas, and Thomas Schwarz, *Efficiently Managing Context Information for Large-Scale Scenarios*, Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications, PERCOM '05, IEEE, 2005, pp. 331–340.

[GD92] K. C. Gowda and E. Diday, *Symbolic Clustering Using a New Similarity Measure*, IEEE Transactions on Systems, Man and Cybernetics **22** (1992), no. 2, 368–378.

[GDR09] Lars Geiger, Frank Dürr, and Kurt Rothermel, *On Contextcast: A Context-Aware Communication Mechanism*, Proceedings of the IEEE International Conference on Communications, ICC '09, IEEE, June 2009, pp. 1–5.

[GDR10] ———, *Aggregation of User Contexts in Context-based Communication*, Proceedings of the 6th Euro-NF Conference on Next Generation Internet, NGI 2010, IEEE, June 2010, pp. 1–8.

[GDR11] ———, *Adaptive Routing in a Contextcast Overlay Network*, IEEE 7th International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2011, IEEE, October 2011, pp. 97–104.

[GMUW08] Héctor García-Molina, Jeffrey D. Ullman, and Jennifer Widom, *Database Systems: The Complete Book*, 2nd ed., Prentice Hall Press, 2008.

[GR95] K. C. Gowda and T. V. Ravi, *Divisive clustering of symbolic objects using the concepts of both similarity and dissimilarity*, Pattern Recognition **28** (1995), no. 8, 1277–1282.

[GSDR09] Lars Geiger, Ronald Schertle, Frank Dürr, and Kurt Rothermel, *Temporal Addressing for Mobile Context-Aware Communication*, Proceedings of the 6th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MobiQuitous 2009, IEEE, July 2009, pp. 1–10.

[HASG07] Mojtaba Hosseini, Dewan Tanvir Ahmed, Shervin Shirmohammadi, and Nicolas D. Georganas, *A Survey of Application-Layer Multicast Protocols*, IEEE Communications Surveys & Tutorials **9** (2007), no. 3, 58–74.

[Hen74] Gary G. Hendrix, *Modeling simultaneous actions and continuous processes*, Artificial Intelligence **4** (1974), no. 3, 145–180.

[Heu02] Dominic Heutelbeck, *Context Spaces — Self-Structuring Distributed Networks for Contextual Messaging and Resource Discovery*, On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science, vol. 2519, Springer-Verlag, November 2002, pp. 248–265.

[Heu05] ———, *Distributed Space Partitioning Trees and their Application in Mobile Computing*, Dissertation, Fernuniversität Hagen, 2005.

[HGM04] Yongqiang Huang and Héctor García-Molina, *Publish/Subscribe in a Mobile Environment*, Wireless Networks **10** (2004), no. 6, 643–652.

[HNP95] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer, *Generalized search trees for database systems*, Proceedings of the 21th International Conference on Very Large Data Bases, VLDB '95, Morgan Kaufmann Publishers Inc., June 1995, pp. 562–573.

[HP98] Z. J. Haas and M. R. Pearlman, *The performance of a new routing protocol for the reconfigurable wireless networks*, Proceedings of the 1998 IEEE International Conference on Communications, ICC '98, IEEE, June 1998, pp. 156–160.

[IN96] Tomasz Imielinski and Julio C. Navas, *GPS-Based Addressing and Routing*, RFC 2009 (Experimental), November 1996.

[IN99] ———, *GPS-Based Geographic Addressing, Routing, and Resource Discovery*, Communications of the ACM **42** (1999), no. 4, 86–92.

[Jac01] Paul Jaccard, *Etude comparative de la distribution florale dans une portion des alpes et du jura*, Impr. Corbaz, 1901.

[JGJ+00] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and Jr. James W. O'Toole, *Overcast: Reliable Multicasting with an Overlay Network*, Proceedings of the 4th conference on Symposium on Operating System Design & Implementation, OSDI'00, USENIX Association, 2000, pp. 14–14.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn, *Data clustering: a review*, ACM Computing Surveys (CSUR) **31** (1999), no. 3, 264–323.

[KCW11] Athanasios Konstantinidis, Antonio Carzaniga, and Alexander L. Wolf, *A Content-Based Publish/Subscribe Matching Algorithm for 2D Spatial Objects*, Middleware 2011, ACM/IFIP/USENIX 2011 International Middleware Conference, Lecture Notes in Computer Science, vol. 7049, Springer-Verlag Berlin Heidelberg, December 2011, pp. 208–227.

[KE13] Alfons Kemper and André Eickler, *Datenbanksysteme: Eine Einführung*, 9th ed., De Gruyter Oldenbourg, 2013.

[KLS07] Aleksandra Kovačević, Nicolas Liebau, and Ralf Steinmetz, *Globase.KOM - A P2P Overlay for Fully Retrievable Location-based Search*, 7th IEEE International Conference on Peer-to-Peer Computing, P2P 2007, IEEE, September 2007, pp. 87–96.

[KR05] Leonard Kaufman and Peter J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley Series in Probability and Statistics, Wiley-Interscience, 2005.

[Lan10] Ralph Lange, *Scalable Management of Trajectories and Context Model Descriptions*, Dissertation, Universität Stuttgart, December 2010, p. 202.

[LCG+09] Ralph Lange, Nazario Cipriani, Lars Geiger, Matthias Großmann, Harald Weinschrott, Andreas Brodt, Matthias Wieland, Stamatia Rizou, and Kurt Rothermel, *Making the World Wide Space Happen: New Challenges for the Nexus Context Platform*, Proceedings of the 7th Annual IEEE International Conference on Pervasive Computing and Communications, PerCom '09, IEEE Computer Society, March 2009, pp. 1–4.

[LCH+07] G. Li, A. Cheung, Sh. Hou, S. Hu, V. Muthusamy, R. Sherafat, A. Wun, H.-A. Jacobsen, and S. Manovski, *Historic Data Access in Publish/Subscribe*, Proceedings of the 2007 inaugural International Conference on Distributed Event-Based Systems, DEBS '07, ACM, 2007, pp. 80–84.

[LDR10] Ralph Lange, Frank Dürr, and Kurt Rothermel, *Indexing Source Descriptions based on Defined Classes*, Proceedings of the 14th International Database Engineering and Applications Symposium (Montreal, QC, Canada), IDEAS '10, ACM, August 2010, pp. 245–256.

[Leo98] Ulf Leonhardt, *Supporting Location-Awareness in Open Distributed Systems*, Ph.D. thesis, Imperial College of Science, Technology and Medicine, University of London, May 1998.

[Leo03] Alexander Leonhardi, *Architektur eines verteilten skalierbaren Lokationsdienstes*, Dissertation, Universität Stuttgart, June 2003.

[LR00] Alexander Leonhardi and Kurt Rothermel, *A Comparison of Protocols for Updating Location Information*, Technical Report 2000/05, Department of Computer Science, Universität Stuttgart, March 2000.

[LR01] Alexander Leonhardi and Kurt Rothermel, *A Comparison of Protocols for Updating Location Information*, Cluster Computing **4** (2001), no. 4, 355–367.

[Mac67] J. MacQueen, *Some methods for classification and analysis of multivariate observations*, Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–197.

[Mal98] Gary Scott Malkin, *RIP Version 2*, RFC 2453 (Standard), November 1998, Updated by RFC 4822.

[MFGB02] Gero Mühl, Ludger Fiege, Felix C. Gärtner, and Alejandro Buchmann, *Evaluating Advanced Routing Algorithms for Content-Based Publish/Subscribe Systems*, Proceedings of the 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, MASCOTS 2002, IEEE, 2002, pp. 167–176.

[Min99] Nelson Minar, *A Survey of the NTP Network*, Unpublished paper, available at http://alumni.media.mit.edu/ nelson/research/ntp-survey99/ [online 2015-11-04], December 1999.

[MJ09] Alberto Montresor and Márk Jelasity, *PeerSim: A scalable P2P simulator*, Proceedings of the IEEE 9th International Conference on Peer-to-Peer Computing, P2P'09, IEEE, September 2009, pp. 99–100.

[MMBK10] D. Mills, J. Martin, J. Burbank, and W. Kasch, *Network Time Protocol Version 4: Protocol and Algorithms Specification*, RFC 5905 (Proposed Standard), June 2010.

[Moo65] Gordon E. Moore, *Cramming more components onto integrated circuits*, Electronics **38** (1965), no. 8, 114–117.

[Moy94] John T. Moy, *Multicast Routing Extensions for OSPF*, Communications of the ACM **37** (1994), no. 8, 61–66, 114.

[Moy98] ———, *OSPF Version 2*, RFC 2328 (Standard), April 1998, Updated by RFC 5709.

[MRR80] John M. McQuillan, Ira Richer, and Eric C. Rosen, *The New Routing Algorithm for the ARPANET*, IEEE Transactions on Communications **28** (1980), no. 5, 711–719.

[Müh01] Gero Mühl, *Generic Constraints for Content-Based Publish/Subscribe*, Cooperative Information Systems, Lecture Notes in Computer Science, vol. 2172, Springer-Verlag Berlin Heidelberg, September 2001, pp. 211–225.

[Müh02] ———, *Large-Scale Content-Based Publish/Subscribe Systems*, Dissertation, TU Darmstadt, 2002.

[Nat97] National Imagery and Mapping Agency, *Department of Defense World Geodetic System 1984 – Its Definition and Relationships With Local Geodetic Systems*, NIMA Technical Report TR8350.2, Geodesy and Geophysics Department, National Imagery and Mapping Agency, July 1997.

[Nav01] Julio Cesar Navas, *Geographic routing in a datagram internetwork*, Ph.D. thesis, Department of Computer Science, Rutgers University, 2001.

[NI97] Julio C. Navas and Tomasz Imielinski, *GeoCast – Geographic Addressing and Routing*, Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking, MobiCom '97, ACM, 1997, pp. 66–76.

[OMM+02] Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani, *Streaming-Data Algorithms For High-Quality Clustering*, Proceedings of the 18th International Conference on Data Engineering, IEEE, March 2002, pp. 685–694.

[OPSS93] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen, *The Information Bus®–An Architecture for Extensible Distributed Systems*, ACM SIGOPS Operating Systems Review **27** (1993), no. 5, 58–68.

[ÖV11] Tamer Özsu and Patrick Valduriez, *Principles of Distributed Database Systems*, Third ed., Computer science, Springer New York Dordrecht Heidelberg London, 2011.

[PB02] Peter R. Pietzuch and Jean M. Bacon, *Hermes: A Distributed Event-Based Middleware Architecture*, Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops, ICDCSW'02, IEEE, 2002, pp. 611–618.

[Pie04] Peter R. Pietzuch, *Hermes: A Scalable Event-Based Middleware*, Ph.D. thesis, Queens' College, University of Cambridge, February 2004.

[Pow96] David Powell, *Group communication*, Communications of the ACM **39** (1996), no. 4, 50–53.

[PSB03] Peter R. Pietzuch, Brian Shand, and Jean Bacon, *Framework for Event Composition in Distributed Systems*, Middleware 2003, ACM/IFIP/USENIX 2003 International Middleware Conference, Lecture Notes in Computer Science, vol. 2672, Springer-Verlag Berlin Heidelberg New York, 2003, pp. 62–82.

[PSB04] Peter R. Pietzuch, Brian Shand, and Jean Bacon, *Composite Event Detection as a Generic Middleware Extension*, IEEE Network **18** (2004), no. 1, 44–55.

[PSVW01] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel, *ALMI: An Application Level Multicast Infrastructure*, Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems, USENIX Association, 2001, pp. 49–60.

[RBB03] Kurt Rothermel, Martin Bauer, and Christian Becker, *Digitale Weltmodelle – Grundlage kontextbezogener Systeme*, Total vernetzt – Szenarien einer informatisierten Welt (Friedemann Mattern, ed.), Xpert.press, Springer-Verlag Berlin Heidelberg, 2003, pp. 123–141.

[RD01] Antony Rowstron and Peter Druschel, *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems*, Middleware 2001, Lecture Notes in Computer Science, vol. 2218, Springer-Verlag Berlin Heidelberg, 2001, pp. 329–350.

[REF+06] Kurt Rothermel, Thomas Ertl, Dieter Fritsch, Paul J. Kühn, Bernhard Mitschang, Engelbert Westkämper, Christian Becker, Dominique Dudkowski, Andreas Gutscher, Christian Hauser, Lamine Jendoubi, Daniela Nicklas, Steffen Volz, and Matthias Wieland, *SFB 627 – Umgebungsmodelle für mobile kontextbezogene Systeme*, Informatik – Forschung und Entwicklung **21** (2006), no. 1, 105–113.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, *A Scalable Content-Addressable Network*, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '01, ACM, 2001, pp. 161–172.

[RHKS01] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker, *Application-Level Multicast Using Content-Addressable Networks*, Networked Group Communication, Lecture Notes in Computer Science, vol. 2233, Springer-Verlag Berlin Heidelberg, October 2001, pp. 14–29.

[Rot03] Jörg Roth, *Semantic Geocast Using a Self-Organizing Infrastructure*, Innovative Internet Community Systems, Lecture Notes in Computer Science, vol. 2877, Springer-Verlag Berlin Heidelberg, 2003, pp. 216–228.

[Rot05] ———, *A Decentralized Location Service Providing Semantic Locations*, Habilitationsschrift, Fachbereich Informatik, FernUniversität Hagen, October 2005.

[SA97] Bill Segall and David Arnold, *Elvin has left the building: A publish/subscribe notification service with quenching*, Proceedings of the 1997 Australian UNIX Users Group Conference, September 1997.

[SMLN+01] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, ACM SIGCOMM Computer Communication Review – Proceedings of the 2001 SIGCOMM conference **31** (2001), no. 4, 149–160.

[SRC84] Jerome H. Saltzer, David P. Reed, and David D. Clark, *End-to-End Arguments in System Design*, ACM Transactions on Computer Systems (TOCS) **2** (1984), no. 4, 277–288 (English).

[TIB08] TIBCO, *TIBCO Rendezvous*, Product Datasheet, 2008.

[Tox14] Bob Toxen, *The NSA and Snowden: Securing the All-seeing Eye*, Communications of the ACM **57** (2014), no. 5, 44–51.

[u-b11] u-blox AG, *MAX-6 u-blox 6 GPS Modules – Data Sheet*, Product Datasheet, September 2011.

[Wax88] Bernard M. Waxman, *Routing of multipoint connections*, IEEE Journal on Selected Areas in Communications **6** (1988), no. 9, 1617–1622.

[WBS+05] Benjamin Weyl, Pedro Brandão, Antonio F. Gómez Skarmeta, Rafael Marin Lopez, Parijat Mishra, Christian Hauser, and Holger Ziemek, *Protecting Privacy of Identities in Federated Operator Environments*, Proceedings of the 14th IST Mobile & Wireless Communications Summit, June 2005, pp. 1–5.

[Wer15] Marius Wernke, *Privacy-aware Sharing of Location Information*, Dissertation, Universität Stuttgart, 2015.

[WL89] D. Jim Walmsley and Gareth J. Lewis, *The Pace of Pedestrian Flows in Cities*, Environment and Behavior **21** (1989), no. 2, 123–150.

[WPD88] David Waitzman, Craig Partridge, and Stephen E. Deering, *Distance Vector Multicast Routing Protocol*, RFC 1075 (Experimental), November 1988.

[WSCY99] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha, *Updating and Querying Databases that Track Mobile Units*, Distributed and Parallel Databases – Special issue on mobile data management and applications **7** (1999), no. 3, 257–387.

[XWI05] Rui Xu and Donald Wunsch II, *Survey of Clustering Algorithms*, IEEE Transactions on Neural Networks **16** (2005), no. 3, 645–678.

[YGM94] Tak W. Yan and Héctor García-Molina, *Index Structures for Selective Dissemination of Information Under the Boolean Model*, ACM Transactions on Database Systems **19** (1994), no. 2, 332–364.

[ZKJ01] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph, *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*, Report No. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, April 2001.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, Proceedings of the 1996 ACM SIGMOD international conference on Management of data, SIGMOD '96, ACM, 1996, pp. 103–114.

Erklärung

Hiermit erkläre ich, dass ich die beigefügte Dissertation selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel genutzt habe.

Ort, Datum	*Unterschrift*