Cerebro: Forming Parallel Internets and Enabling Ultra-Local Economies

by

Polychronis Panagiotis Ypodimatopoulos

Bachelor of Science in Computer Science
Aristotle University, 1999
Master of Science in Enterprise Information Systems
King's College London, 2000

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2008

Author_____
Program in Media Arts and Sciences
August 8, 2008

Certified by_____
David P. Reed
Adjunct Professor
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by_____
Deb Roy
Chair, Academic Program in Media Arts and Sciences
Program in Media Arts and Sciences

Cerebro: Forming Parallel Internets and Enabling Ultra-Local Economies

by

Polychronis Panagiotis Ypodimatopoulos

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on August 8, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

Internet-based mobile communications have been increasing rapidly [5], yet there is little or no progress in platforms that enable applications for discovery, context-awareness and sharing of data and services in a peer-wise manner among collections of devices in the same physical area. This is important because proximate devices may need to communicate directly when no infrastructure is available, and because such local access may be an efficient alternative to connecting a large number of sensors, effectors, and people to a readily accessible, universal central system.

This thesis presents the design, implementation and evaluation of Cerebro, a system that allows suitably equipped humans and objects in the same physical area to discover each other and share data and services. Cerebro offers two basic services: a presence service that propagates information about local devices through an automatically generated mesh network and an innovative data transport service to transfer data via this network. On top of these services, Cerebro offers an extensible Application Programming Interface (API). In a mobile mesh network with N devices, Cerebro offers an upper bound of $O(N)$ on traffic overhead to maintain presence information at any part of the network and responsiveness to device arrival/departure events that take at most $O(N)$ time to propagate throughout the network. This makes Cerebro a scalable and useful addition to mobile service delivery.

Thesis Supervisor: David P. Reed
Adjunct Professor
Program in Media Arts and Sciences

Cerebro: Forming Parallel Internets and Enabling Ultra-Local Economies

Polychronis Panagiotis Ypodimatopoulos

The following people served as readers for this thesis:

Thesis Reader_____

Andrew Lippman
Senior Research Scientist
Program in Media Arts and Sciences

Thesis Reader_____

Walter Bender
Senior Research Scientist
Program in Media Arts and Sciences

Thesis Reader_____

William J. Mitchell
Alexander Dreyfoos Professor of Architecture and Media Arts and Sciences
Program in Media Arts and Sciences

Acknowledgements

To my advisers, David P. Reed and Andy Lippman, I would like to express my deep gratitude for their critical insights and guidance that helped shape the core of this thesis. I would also like to thank my readers, Walter Bender and Bill Mitchell, for their precious time and unconventional ideas. Working with all four of you has been an exciting experience and an amazing privilege.

Of course, many thanks and love go to my wife, Bettina, for her help and support. First, you supported me through some quite hard times, including my application to the Media Lab and then you 've been patient and supportive while I've been there. Thank you for making my dream come true.

One of the reasons for joining the Media Lab was because here, "thinking outside the box" is the rule rather than the exception. Thank you for giving me the opportunity to join such an inspirational and fun place. Two years later, I still feel like a kid in a candy store everytime I enter the building. This "salon de refusé for electrical engineers" is also a playground for innovation through self-expression.

Equally amazing is the OLPC project. Not only for its beautiful purpose, but also for its technical breakthroughs. I deeply thank OLPC and its wonderful community for providing the necessary resources and feedback that made critical parts of this thesis possible. Special thanks go to Michail Bletsas, whose ideas helped kick off this thesis, Michael Stone, Chris Ball, Scott Ananian, Kim Quirk and many others.

Also, I would like to thank Jhonatan Rotberg and Deb Roy for

brainstorming with me and for helping me gain better perspective on Cerebro's applications.

Special thanks to Rene De Santiago and Noah Silverman for their contributions to Cerebro as part of their UROP.

You thought I'd forget about you? Many thanks to my lab-mates David, Dawei, Fulu, Grace, Hector, Kwan, Nadav and Sung for their support and for making this program fun. May the network be with you.

Isn't it? so, look for a puzzle somewhere else.

"It does not matter what you do, as long as you are the best at it."
For my father, Takis

Contents

List of Figures

Chapter 1

Overview and Motivation

What would the internet look like today, if suddenly our internet connection went back down to 1200 baud[1]? YouTube[2] would probably be useless, but would the internet as a whole be rendered useless or would some services and applications still be usable? This was actually the case about 30 years ago when some people were sending emails as part of their everyday work and fewer were only dreaming about watching video over the internet. It could also be argued that it was the instant-on nature of broadband, not its increased bandwidth that had a large impact on internet application development and usage.

However, not all contemporary network applications require high bandwidth; using and contributing to Twitter[3] [15] and other similar social networking services [48] require little bandwidth, but provide disproportionally large utility by aggregating and redistributing the information that each user contributes. Even email and search engines —two cornerstone internet services— have relatively insignificant bandwidth requirements on the user's side.

We believe that in the future more applications and services will emerge that draw their value from exchanging small pieces of information with large numbers of users, rather than acquiring large quantities of data from a single

[1] "Baud" is the data rate unit for symbols per second.
[2] Online user videos: http://www.youtube.com
[3] Allows users to broadcast a short text message about their current activity: http://www.twitter.com

source. Social networking platforms and services, such as Twitter, are an excellent example of this trend. Even when watching a friend's newly posted photographs on Facebook[4], the value does not come from being able to access the raw bits of the photograph—Facebook could have pointed to another location where the photographs are stored—, but rather from getting an alert that aggregates all our friends that have posted new photographs recently. Of course, the alert itself has negligible data size, but gathering various alerts from a multitude of sources provides unprecedented value to the user.

One could expect such applications and services that exchange low-volume, but high-value data with numerous other users to become more pervasive and ubiquitous in the user's everyday life, offering more than just social networking services. This trend is further strengthened by the fact that the density of modern urban environments in combination with people's tendency to share increasingly more personal information [37][29] create a dynamic environment where information is constantly created and shared with the user's environment.

This dynamic environment could be well-modeled by mobile mesh networking which allows mobile devices to communicate directly with one another, without relying on any infrastructure that is external to the mesh network itself. As such, mobile mesh networking could be key in providing ubiquitous, ad-hoc communication in future urban environments. Still, mobile mesh networking is usually dismissed on the basis that a mesh network's capacity does not scale [20] enough to satisfy the bandwidth requirements of today's internet users, or that there is no viable business model behind it.

1.1 Cerebro

This thesis attempts to address both of the above claims in order to bridge the gap between the current state of affairs in mobile communications and alternative business models that could arise in the future. We present Cerebro, a system that allows humans and objects in the same physical area to discover each other and share data and services. Cerebro brings closer a

[4] http://www.facebook.com

future where humans will be able to discover and interact with their physical environment as part of a mobile network that could span thousands of participants.

We envision a future where digital communication takes place not only through third parties such as Internet Service Providers (ISP) and mobile operators, but also directly between users and between users and businesses by means of a mesh network that they form using Cerebro. This type of communication is pertinent to the physical location of the communicating parties. Our motive for pursuing this future is not to avoid having to pay fees to ISPs and mobile operators but rather to create an architecture that effectively addresses discovery, context-awareness and communication between entities that lie in the same physical area. Such an architecture is actually further strengthened by the presence of an internet service, not merely to provide an internet service to Cerebro's users, but to connect mesh networks together across different geographical locations and to make available over the internet the information content that is created within each mesh network. We believe that the value is no longer in accessing the internet from the mesh network, but in locating and accessing information within a mesh network from the internet.

Cerebro is a service that encompasses the necessary protocols and services to create a mesh network out of mobile devices, within which each device can modify and share its profile along with any bulk data that may accompany it. Cerebro provides two basic services: a presence service that provides information about all devices that are accessible within the mesh network that Cerebro creates and a data transport service to transfer data among those devices. Finally, Cerebro offers and extensible application programming interface (API) to allow applications to be built on top of the services that it offers.

Cerebro is free software. As of this writing it can be found online at `http://cerebro.mit.edu/`

1.1.1 Stakeholders in Cerebro's Network

Cerebro affects mainly mobile users, especially in dense environments where rich information content is created simply by the dense presence of users. By providing an architecture that allows users to digitize their presence and profile and communicate efficiently, we break ground for business models that draw on the physical presence and profile the each user shares and her interaction with the rest of the network.

1.2 Technical Challenges

If we were to put together a technical solution based on existing technologies for service discovery and data transport in a mobile mesh network, we would end up with an architecture that does not scale because existing service discovery mechanisms do not perform well in a wireless and mobile and environment and existing routing protocols that create a mobile mesh network do not offer an embedded discovery mechanism.

Also, one-to-many reliable data transfer in a wireless setting is currently inefficient because the amount of repetition in a data transmission is proportional to the number of receivers. Ideally, we would like to only send each piece of data once, regardless of the number of concurrent wireless receivers.

Cerebro approaches these challenges by combining concepts from the areas of routing in Mobile Ad-hoc Networks (MANETs) and Sensor Networks (SN). Traditionally, routing protocols for MANETs and data transport protocols have been considered as two separate domains: The first deals primarily with the aspects of routing that keep the network connected, while the second is concerned with transferring data from some source to some destination.

Cerebro blends aspects from both domains in order to maximize their utility. We can summarize how data transport and routing protocols can benefit from each other in a mobile mesh setting as follows:

- If you want to send data to a bunch of destinations, you better know how they are layed out in order to make the best out of the wireless medium in terms of traffic overhead.
- If you want to have information about as many nodes as possible in order to inform your routing decisions, each node should share its profile as efficiently as possible in terms of traffic overhead.

By "traffic overhead" we imply not only data retransmissions, but the total amount of data transmitted pertaining to the specific data transfer over the actual size of the data to be transferred. This fraction is usually referred to as "stretch". Ideally, "stretch" should be equal to 1 in the case of one sender and multiple wireless receivers —all being in range with the sender—, regardless of the number of receivers. For TCP-based data transfer, "stretch" is proportional to the number of receivers because the data must be sent at least as many times, as there are receivers.

The first clause suggests that network layout information that is gained from a proactive routing protocol can improve the performance of a data transport protocol operating in this network.

The second clause suggests that the more efficient the data transport protocol is, the more information we obtain about data and services that nodes share in the network and the more informed decisions we can make about which node we should contact first and how to reach it.

1.3 Contributions

This thesis' contribution is three-fold. First, we demonstrate an extensible architecture built around Cerebro's vision that allows us to revisit and provide innovative solutions for calculating, organizing and sharing the presence of mobile users and objects in physical proximity. Second, we implement an innovative mechanisms to reliably transfer data to multiple wireless receivers in parallel and to adjust the transmission frequency of presence beacons. Third, we provide an account of applications and potential

business models that can be built around the network of users that Cerebro provides.

1.3.1 Architectural Contribution

We designed and implemented Cerebro having in mind the inherent but unsatisfied human need to seamlessly share, discover and access information that is available but uncaptured in a mobile, ad-hoc setting (e.g. Is a friend nearby? Anyone else speaks English around me in this foreign country?). Cerebro provides an architecture through which it is straightforward not only to capture this information by representing it effectively, but also to distribute it efficiently to all network participants.

Cerebro departs from the server-centric approach where information is centrally stored and distributed thus inhibiting direct user-to-user interaction and community formation. Instead, we enable individuals to make their presence directly known to their environment, including any information they wish to pass along, thus lifting issues of server trust and security. Our priority is to keep the user —and her mobile device thereof— at the center of all interaction with other network participants, instead of placing a server at the center of all communications. We can then propose new business models that arise from this architecture, rather than merely expand legacy system architectures to sustain existing business models.

1.3.2 Performance Contribution

Cerebro's design and implementation not only satisfy the above objectives, but also do so by exhibiting well-defined scaling and performance properties. Cerebro implements a breakthrough in data transport and dissemination of presence information in mobile mesh networks: Data sharing with neighbors takes almost[5] constant time (see also section 6.1.1) regardless of the number of destinations, while the traffic overhead for dissemination of presence

[5] Time-to-completion is constant for about 90% of the data to be sent, while section 7.1 proposes some techniques that deal with the remaining 10%.

information is linear with the number of nodes regardless of the layout of the network.

To the best of our knowledge, Cerebro's presence mechanism is the only one to offer a traffic overhead in any part of the mesh network that is linear with the total number of nodes in the network.

1.3.3 Forming "Parallel Internets" and Enabling Ultra-Local Economies

It is not immediately evident how a business model can be built on a network where there is no single entity in control of all network traffic. Instead of treating a mobile mesh network as a mere extension of the internet backbone, we focus our efforts on providing context-awareness, user and service discovery, communication and group establishment in small physical areas. MANETs and SNs are traditionally used to get data in and out of the mesh network, whereas Cerebro powers applications whose goal is to stay in the mesh network rather than to get out of it.

The resulting mesh network that is built using Cerebro is a miniature of the internet in that each node in the mesh network is analogous to an autonomous system (AS) in the internet. The rules that govern the relationship among different autonomous systems in the internet and nodes in a mesh network are similar; the default goal of an AS is to maximize its profit and utility —as experienced by the users that live within the AS—, while minimizing the network traffic and monetary cost of communicating with neighboring autonomous systems. Therefore, the formation and spatial distribution of multiple, disjoint mesh networks hints the term "Parallel Internets".

By acting autonomously and using Cerebro's presence and data transport protocols, each node can share or trade information and services with other nodes in its physical vicinity, thus forming an "Ultra-Local Economy".

1.4 Thesis Structure

This thesis comprises seven chapters, this introduction being Chapter 1.

Chapter 2 develops terminology and establishes related work from a technical perspective.

Chapter 3 starts by presenting earlier designs and experiments that motivated Cerebro, then provides a detailed technical and theoretical description of Cerebro's system architecture and design goals. Chapter 3 closes by exploring the theoretical limits of Cerebro's presence and data transport protocols.

Chapter 4 provides a description of Cerebro's implementation and the application programming interface (API) that Cerebro offers.

Chapter 5 provides an account of applications and potential business models that draw upon and extend Cerebro's functionality.

In Chapter 6 we evaluate Cerebro's performance by providing experimental results on a testbed with tens of nodes.

Finally, Chapter 7 concludes by discussing Cerebro's protocols and respective implementation's strengths and weaknesses and by presenting future work that could further increase Cerebro's scalability and performance.

Chapter 2

Related work

As mentioned in the Introduction, this thesis' technical contributions mainly involve:

- an extensible architecture that allows us to provide innovative solutions for calculating, organizing and sharing the presence of mobile users and objects in physical proximity
- mechanisms to reliably transfer data to multiple wireless receivers in parallel and to keep the network traffic overhead for presence information linear with the total number of nodes in the network

This chapter starts by establishing the terminology that is used throughout this document and then presents related work to Cerebro's key contributions.

2.1 Terminology

In order to set some common ground for all technical descriptions in this thesis, we provide a list of technical terms that are commonly used.

mesh network - A mesh network —or simply "mesh"— is a wireless, usually mobile, network of devices that is self-configurable and in which devices can communicate directly or through other devices that act as

bridges. In the literature, a mobile mesh network is also referred to as MANET.

presence information - is the information that signifies user state in some networked application such as instant messaging and IRC[1]. The XMPP standard [82] describes an extensible set of user states such as "away", "do not disturb", explicitly expressing the user's ability and/or willingness to communicate with others. Each user maintains a contact list and contributes her own presence to a server that maintains presence information for all users. The server feeds presence information back to the user about all members on her contact list.

presence beacon - A data frame periodically sent by every node, which contains a list of presence entries that the node has information about either directly, or indirectly through a neighbor.

presence entry - Carries information about some remote node. A presence entry consists of four fields: the ID of the remote node, the ID of neighbor where this presence entry originated from (also known as "the witness"), the distance to the remote node as reported by the witness and a sequence number to break cycles.

presence table - Holds information about all nodes that we received presence entries about. The presence table contains the following fields: the ID of the remote node, the IDs of the first two nodes along the path towards the remote node, the total distance to the remote node, a list of timestamps when there was a presence entry arrival from the remote node along the same path and a time estimate until which we can wait for a presence entry arrival, before we declare the remote node as offline.

node profile - A extensible list of key/value pairs each of which provides some piece of information about the user (such as name, age, groups the user belongs to) and the device (such as the services that this device offers).

network interface - The hardware radio used to communicate with other devices, such as WiFi, Bluetooth, ZigBee, etc.

[1] Internet Engineering Task Force RFC-1459

2.2 Architecture

2.2.1 Hierarchical vs. Ad-hoc Architectures

It could be argued that internet's hierarchical architecture is inherently unsuitable for the ad-hoc, intermittent connections between people that are in physical proximity. Internet-based architectures fail to address communications in physical proximity because they were designed with a different goal in mind: scalable interconnection of arbitrary types of networks at potentially global scale. Furthermore, mobile operators enable people to communicate while they are mobile, but not directly with each other as this would pose a direct threat to their existing business model.

On the other hand, it has been hard to adopt alternative architectures, such as mesh networking. Zhang et al. [81] show that scalability in large mobile ad-hoc networks is hard to address due to the combination of long routes forming in the network and node mobility. Additionally, Balakrishnan [20] shows that the network capacity in a mesh network is inversely proportionate to the square root of the number of nodes N. Therefore, on average, the wireless network's nominal capacity should be divided by $\sqrt{N}$ just in order to form the mesh network.

2.2.2 Current Trends in Presence and Mobile Social Networking Architectures

Wikicity [70] is an attempt to build a real-time map of a city showing where all the action is taking place. This is done by centrally aggregating data from cell phone users, taxis and buses and therefore the system is based on some infrastructure. Roofnet [13] connects different users together by forming a mesh network, but the users are not mobile (an antenna is fixed at the roof of user's house). Biomap [54] is a system that provides mobile users information about the emotional arousal the people have on average in the present area. Participating users upload their emotional status and physical location to a central server from which information is fed to future visitors of that location. Again, this is a centralized system that covers only one aspect

of our proposed system (sharing emotional data). Probably the platform that comes closer than anyone else is the mesh network of the XO laptop by OLPC [55]. However, the presence mechanism that is used for discovering other users in the mesh network is based on each user broadcasting her presence to the whole mesh network, contrary to our approach that only broadcasts presence information to immediate neighbors and therefore exhibits better scalability properties.

The Table 2.1 summarizes similar efforts in the area of mobile and/or social networking and how Cerebro differs from them. The following criteria are examined:

- Presence in physical space: maintaining information about other users that are in physical proximity
- Mobility: being able to run the application on a portable mobile device
- Extensible: provide a programming API to allow users to write their own applications on top of this platform
- no internet connection: an internet connection is not required for normal operation
- no mobile operator: connection to a mobile operator is not required for normal operation
- chat: users can chat with each other directly
- file exchange: users can exchange files in a peer-to-peer fashion

Rakket is allows users to discover each other on a local network subnet. Typically, an internet connection is not required, but a way to disseminate unique IP addresses in the same subnet is required, which can be very hard to accomplish in a mobile environment. Meetro is an interesting, Web-2.0, presence application that identifies the location of each user based their IP address. As a result an internet connection is required. Similarly, Jambo requires connection to a mobile operator, Jaiku and Twitter (also Web-2.0 applications) require an internet connection. Additionally, Jaiku and Twitter do not provide a way for users to interact with each other. Facebook is a well

known Web-2.0 application that again requires internet connection. Mobile IP is an IETF standard that allows mobile users to be reachable at the same IP address and as such is irrelevant to providing presence information among users, or enabling social interaction. Finally, Location-based services are services that mobile users can have through their mobile operator, so a form of infrastructure is always required.

	Presence in physical space	Mobility	Extensible	no internet connection	no mobile operator	Ref.
Rakket	yes	no	no	yes	yes	[11]
Meetro	yes	no	no	no	yes	[8]
Jambo	yes	yes	no	no	no	[7]
Jaiku	no	yes	no	no	yes	[6]
Twitter	no	yes	yes	no	yes	[15]
Facebook	no	yes	yes	no	yes	
Mobile IP	no	yes	no	no	yes	[65]
Location-based services	yes	yes	no	yes	no	[77]
Context-based routing	no	yes	no	yes	yes	[3]
Cerebro	yes	yes	yes	yes	yes	

Table 2.1: Systems and applications that provide Presence and/or Social Networking.

2.3 Presence and Data Transport Mechanisms

2.3.1 Presence Information and Service Discovery

There are different protocols to provide presence information and to discover services over some type of network. Bonjour [2] and Avahi [1] are both implementations of Zeroconf [16], a distributed protocol to establish unique IP addresses without the need for centralized services like DHCP [28] and DNS [50]. In addition to establishing unique IPs, both Bonjour and Avahi locate devices such as printers and the services that those devices offer on a local network using multicast Domain Name System (mDNS) [9] service records. Multicast DNS implements a distributed DNS service according to which devices use a multicast IP address (usually it is the address `224.0.0.251`) to advertise and discover names and services.

mDNS is at the core of Zeroconf's discovery mechanism and, according to mDNS, it should be used "in a small network". However there is no clear definition for the upper limit of the size of the network. Indeed, large deployments of devices where discovery is based on the mDNS protocol [56] report that it does not scale above 20 devices [57] in a mesh network.

2.3.2 Routing in Mobile Ad-hoc Networks (MANETs)

We are interested in routing protocols in MANETs because one could base a presence information mechanism on a routing protocol, given that the protocol would be suitable to be used in a mobile setting and would have low overhead.

Routing protocols in MANETs are primarily concerned with the establishment of paths along which data is forwarded from some source to some destination. The paths are stored in a routing table and are used whenever traffic needs to be forwarded to some remote node. The proactive class of routing protocols attempt to discover all nodes that are reachable in the network and the respective shortest path to them. As a result, they provide a basic form of presence about other nodes that also includes a distance metric that is pertinent to the protocol.

The proactive protocols are further broken down into the "link-state" and "distance-vector" protocols [74]. The former protocols, such as OLSR [24], generally require that all nodes communicate their routing table to all other nodes in the network, so that each node can construct a global picture about the network and make an optimal decision about on how to affect its routing table. However, this yields high traffic overhead that increases significantly in a mobile network because routing table changes happen more often.

"Distance-vector" protocols only forward routing table changes to immediate neighbors and therefore their traffic overhead is lower. On the downside, they have traditionally suffered from issues such as the "count-to-infinity" problem [74]. DSDV [66] presented a solution to this problem, and is therefore an attractive solution to constructing a list of all nodes present in the network because of its relatively low overhead. Still, DSDV is

event-driven and tends to create excessive traffic that is driven by topology changes that, in a mobile network, may be frequent.

2.3.3 Data transport protocols in mesh networks

2.3.4 Deficiency of TCP/IP in mobile/wireless networks

Efficient data transport is a critical component in a mesh network architecture. TCP [74] is the de-facto standard for reliable data transfer in packet-switched networks. However, TCP was designed to offer data transport reliability between two hosts, whereas in our environment it is often the case that one wireless node needs to send data to multiple wireless neighbors. For example, a teacher attempts to send a document to all classroom participants, or a device joins a mesh network and needs send its profile to all other devices. In these cases, TCP would require a separate session with each destination sequentially. However, this would be a waste of bandwidth because the same data would need to be transferred as many times as there are destinations, which loses the broadcast advantage of radio. This is similar to a data multicasting problem [85], but we focus on optimizing for the case of having multiple, direct wireless neighbors forming a fully connected network. To the best of our knowledge, existing approaches would require multiple, separate sessions in order to achieve reliable transport. Instead, Cerebro attempts to minimize both intra- and inter-flow repetition of data, which represent retransmission of frames and whole files respectively.

The broadcast nature of radio has been exploited extensively when using network coding in order to combine the information content of two or more packets into a single one [44] [32]. However, these protocols focus on increasing throughput, rather than providing reliable data transport.

Reliable data transport in Sensor Networks (SN)

There is extensive literature around data transport protocols in the area of SN. SNs are mainly used to deploy program code from some node —usually

referred to as the "sink"— to all sensors and then gather data from all sensors back at the sink.

[62], [45], [43], [83], [36], [73] provide reliable data transport by specializing in many-to-one transfers. However, like in TCP's case, they do not perform parallel data transfer by making direct use of the broadcast nature of radio.

Trickle [46] is another seminal protocol in SNs that achieves eventual data consistency. Although it has only been tested with relatively small data sizes (around tens of bytes), Trickle is efficient mainly because it is density-aware, up to some constant factor. In this respect, it makes use of the broadcast advantage of radio by silencing all neighbors that do not have a newer version of the data at hand, while they can all receive a new version of the data when broadcasted by some neighbor.

However, Trickle does not provide a mechanism to ensure consistency for large data that span multiple packets. Also, the protocol requires that a directed tree is already formed and will be used to signal where new versions should arrive from, since Trickle does not maintain a history of previous versions of the data. Finally, the protocol is adaptive to a varying network density only up to some constant and this value must provided a-priori to fine-tune the protocol's behavior.

Chapter 3

System Design

This chapter describes Cerebro's design, the algorithms and protocols that it introduces and early designs that provided useful lessons for Cerebro's implementation. Cerebro is an ongoing project, phased for adoption by One Laptop Per Child (OLPC) and as such it is expected to further evolve in the future.

3.1 Early designs: Teleporter

Teleporter, now an essential part of Cerebro's architecture, was initially built to perform file reconciliation among a group of mobile devices. The goal was for two or more devices to synchronize a potentially disjoint set of files while in range with each other. Teleporter involved a basic discovery mechanism that would trigger opportunistic and connectionless data transfers among the devices. Teleporter, which provides both a "beacon-based" discovery service and an efficient data transport, is an essential part of Cerebro's architecture.

The initial version of Teleporter was demonstrated using three wireless devices placed at different locations in the 5-story building housing the MIT Media Lab. Device "A" was placed in the basement, device "B" was on the fourth floor and "C" was in the elevator, as shown in Figure 3-1. "A" was equipped with a camera and was out of range from device "B". Device "A"

was regularly taking new pictures and “C”, the device in the elevator, would carry the pictures over to device “B” on the fourth floor. However, “C” was only in range with either “A” or “B”, but not both and only during the time that the elevator’s doors were open at the basement or fourth floor respectively. Elevator doors generally remained opened for about 5 seconds for ingress/egress during which time devices had to discover each other and exchange pictures. The idea was that using regular discovery protocols such as Avahi [1] and Bonjour [2] it would take several seconds for devices to discover each other and establish unique IP addresses. This would be because those protocols do not adapt to dense networks and as such they advertise local services as infrequently as possible, thus increasing the average time to discover a service. Furthermore, in order to establish a unique IP address, each device must compare its IP address with everyone else’s, although this was not a significant problem in our demo because the total number of devices was small.

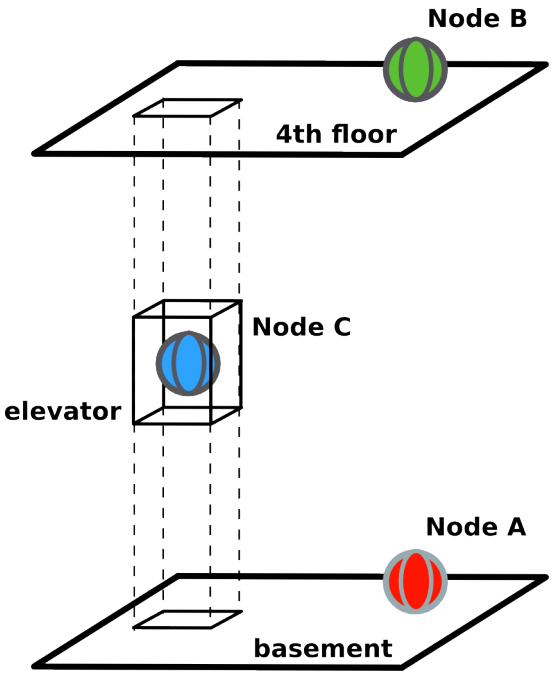


Figure 3-1: Layout of the three nodes that demonstrated an early version of Teleporter.

Teleporter did not require the IP protocol altogether and had a primitive mechanism to adapt its rate of beacon advertisements. As a result, the discovery process would take less than a second (the reception of a beacon

would suffice; beacons were transmitted every second, so the average reception time was less than that), allowing plenty of time to transfer a 500KB picture over a 10Mbit wireless link.

For the rest of this chapter, we present the presence mechanism that provides the presence information, then we describe the collaboration mechanism that builds on top of the presence mechanism by structuring presence information and establishing an efficient data transport mechanism.

3.2 The Presence Mechanism

Traditionally, "presence" [82] is the information that signifies user state in some networked application such as instant messaging and IRC[1]. The XMPP standard [82] describes an extensible set of user states such as "away", "do not disturb", explicitly expressing the user's ability and/or willingness to communicate with others. Each user maintains a contact list and contributes her own presence to a server that maintains presence information for all users. The server feeds presence information back to the user about all members on her contact list.

In this section we analyze the way we extend the current notion of "presence" to include more comprehensive information that is better suited for a mesh network, then we present the protocol that provides presence information and analyze its theoretical limits. The intuition behind this protocol is simple: If necessary, we decrease the freshness of the presence information to accommodate increasing numbers of nodes, while keeping the cumulative overhead in the network linear with the total number of nodes.

3.2.1 Presence Information

Presence information is the epicenter of Cerebro's functionality. Providing scalable presence information about a highly dynamic and mobile swarm of nodes has been a tough goal for similar efforts in the past [75] [34].

[1] Internet Engineering Task Force RFC-1459

This thesis focuses on a set of potentially mobile nodes that form a network by being present in the same physical area. Humans need not necessarily be attending each node. As a result, we extend the notion of "presence" to include meta-information about the state of each node as a whole, rather than just the user operating some application. We organize the meta-information pertaining to each node by looking at the OSI model [60] for networked hosts. We then attempt to exploit the physical proximity between the nodes by sharing as much information with them as possible.

Since the nodes that form the network are in the same physical area, the first piece of information we are interested in is a list of all nodes participating in the network. As this is a potentially mobile and dynamic network, we need to ensure that this list is as up-to-date as possible. Additionally, we attempt to provide information on the physical layout of the network and a sense of distance between the nodes. We are not necessarily interested in providing accuracy in the layout or the absolute location of other nodes, which would be the objective of a localization service, such as GPS. For example, when communicating with other users on a street or in a mall, we are probably not interested in the actual geographical location of the other users, but rather in coarse distance predicates such as "next to me", "same building", "same neighborhood". We further extend "presence" to include state information about the applications running at each node and, of course, information about the user herself.

Table 3.1 summarizes our extensions to traditional "presence" in conjunction with the OSI layer model. According to OSI, each networked host implements a number of layers each of which abstracts the complexity of the layers below it, while offering a specific service to the layers above it. As a result, it is possible to build network applications without knowledge of the implementation of the underlying network infrastructure. The actual OSI model does not include a "User" layer, but we added it in the table in order to differentiate between presence information about the user and meta-information about the application itself.

XMPP is currently the de-facto standard for presence, providing extensible

information such as user status, avatar, etc. At the *user layer*, Cerebro offers similar information with the addition of any type of file that can be part of the user's profile. A user avatar is a special type of file, usually a picture representing the user. However, there really is no reason why the user cannot choose to be represented by a sound, or some other type of media. In the case of OLPC, it could be the XO colors or even a unique SVG file per user. In the future, Cerebro's user profiles should be compatible with the XMPP standard to ensure interoperability with existing applications that leverage presence information.

To the best of our knowledge, "presence" is not used to represent any information about the application itself, or any other layer below the application layer thereof. This is actually no surprise because current applications that employ presence information are primarily used for communication over the internet where physical presence is substituted by virtual presence in order to provide a more compelling communication experience. Providing any information as part of user's presence about any layer below the application itself would defeat the whole purpose of abstracting the complexity of the underlying network by using layers. However, in Cerebro's case we adopt a cross-layer approach [17] in which information from any layer could inform a user's actions and the application's behavior.

We include application state and capabilities in our presence information to notify other users about the applications we are currently sharing or are willing to share and their respective capabilities. For example, we may be willing to edit a document collaboratively or engage in a chat session with a friend or stranger [40]. In another application example, an unattended node can offer temporary storage and processing power to nearby nodes and advertises this service as part of its presence information (see also Section 3.3.1). Finally, users can develop their own applications using Cerebro, so having the ability to state the capabilities of their applications is crucial in order to initiate any useful interaction with other users. Applications are discussed in more depth in Chapter 5.

At the *network layer*, we extend presence to include information such as the

fact that a node may have an internet connection and that the user is willing to share with others, the terms on which she is willing to share an internet or other similar service, whether the node is willing to route and propagate information for other nodes and which nodes specifically, etc.

Information from the *link layer* that we include in a node's presence involves its own inference of the state of other nodes (for node state inference, see also Section 3.2.9) and its distance from them (see also Section 3.2.4). By receiving distance measurements from other nodes as part of their presence information, we can construct the network's connectivity graph as shown in Section 5.1.

Finally, the *physical layer* also has some significant information to contribute to a node's presence. We use information from this layer to notify other nodes about the availability of different radio types. We would expect users to transition communication from one type of radio to another based on different trade-offs such as cost and power usage vs. bitrate. For example, nodes may be using ZigBee [42] to share presence information on a low power budget, while transitioning to WiFi, if available, to exchange a large file. Similarly to the *network layer*, a user may also choose to share a GSM service with others based on criteria that she sets.

OSI Layer	Traditional presence information	Cerebro-based presence information
User	User state, avatar, etc.	Extensible user profile
Application	N/A	Application state and capabilities
Network	N/A	Internet connectivity, network capabilities and related offered services
Link	N/A	Reachability, list of nodes/neighbors, network layout
Physical	N/A	Radio types available (eg. WiFi, Bluetooth, ZigBee, GSM, etc)

Table 3.1: Traditional vs. Cerebro-based presence information sorted by OSI layer.

An example of an node's profile is shown in Figure 3-5.

3.2.2 Algorithm Requirements

Given a mesh network, we wish to provide every node with presence information about every other node in the network. We need to account for

the following network properties and/or requirements:

mobility - nodes may be mobile at a different degree — node mobility may vary from low (eg. a person walking on the street) to high (eg. a car driving on the street) and we need to accommodate them all as much as possible.

dynamicity - dynamicity stands for the rate at which nodes enter or exit the mesh network for various reasons. This property correlates with mobility to some extent, but may well apply to a static network with hundreds of nodes because the varying radio link quality causes remote nodes to appear as if they constantly leave and join the mesh network.

responsiveness - we would like to maximize the algorithm's responsiveness to various events, such as a remote node leaving the network or changing its presence information. The more dynamic and mobile the environment is, the less responsive the algorithm will be, or the more bandwidth it will require to keep up with the changes.

bandwidth limitations - Per-node presence information may vary from a few bytes to several megabytes, if it includes a file for example. On the other hand, bandwidth is probably the most scarce resource in a mesh network. We must strike a balance between the amount of information we need to share and the available bandwidth, while doing any necessary data transfer as efficiently as possible.

processing power - We aim for nodes that can be highly mobile and can operate on a low power budget. Most of the time this translates to having low processing power available at each node. We need to minimize algorithm complexity in order to keep CPU usage low even for large-scale networks.

power budget - Requiring low power is always important. However, we assume that meeting the previous two requirements will also ensure operation on a low power budget, as CPU and radio usage account for most of the power consumed on a mobile device.

One way to distribute presence information is to have each node periodically broadcast its own presence and then re-broadcast any presence it may receive from its neighbors. Given infinite bandwidth, this would probably be the fastest way to propagate presence information throughout the network. However, serious bandwidth and power limitations make such a scheme prohibitive. This is actually one of the initial approaches that OLPC had adopted for presence and discovery in an ad-hoc environment that combined mDNS [9] and the 802.11s networking standard and resulted in network capacity saturation [56] with as few as 10 nodes because mDNS would blindly broadcast presence information and flood the network.

In designing an efficient algorithm for dissemination of presence information, we draw intuition mainly from the field of *Sensor Networking* and routing in *Mobile Ad-hoc Networks* (MANETs).

Routing algorithms for MANETs fall mainly into two categories: *Reactive algorithms* and *Proactive algorithms* [49]. The most basic type of presence information we could provide is a mere list of all nodes in a mesh network. Proactive routing algorithms already provide this basic information, as well as a route to each node. We choose to base our algorithm that provides presence information on a subclass of proactive algorithms called *distance-vector routing algorithms* because they seem to provide the basic information that we need about other nodes, while making no assumptions about the layout, mobility and size of the network.

But, why are we considering routing algorithms since what we need to provide is presence information? Indeed, in the early steps of this work we did not consider routing algorithms at all. Cerebro's presence protocol was designed from scratch in search of the optimal way to establish presence information in a mesh network. However, the resulting algorithm turns out to be a distance-vector routing algorithm for MANETs. As a result, it is possible to use Cerebro both for establishing presence information and routing in the mesh network.

3.2.3 Algorithm Description

We assume a set of nodes forming a mesh network. Every node broadcasts a Presence Beacon to its one-hop neighbors, who we will now refer to simply as "neighbors". A Presence Beacon is transmitted every τ seconds. During this time, a node receives and processes beacons from its neighbors. At the end of this period, the node forms a new beacon that it broadcasts to its neighbors. Each beacon is a list of Presence Entries and each Entry contains four fields:

1. target node that this Presence Entry refers to
2. witness of the current Presence Entry (see also Section 3.2.8)
3. reported distance to the target (see also Section 3.2.4)
4. sequence number of this Presence Entry (see also Section 3.2.5)

If a node does not receive any beacons during some period τ, it creates a beacon with a single Presence Entry about itself: The target and witness are set to itself, the sequence number is increased by 1 and the distance is set to 0. A node uses the Presence Beacons that it receives from its neighbors to measure the distance from its neighbors to itself. Then, it parses the Presence Beacon for valid entries which it uses to adjust its Presence Table about remote nodes that its neighbor is reporting about.

3.2.4 The Distance Metric

Cerebro treats the wireless link between any pair of nodes as being asymmetric, that is, the quality of the link is not considered the same in both directions. We base the distance metric D on previous work done in [25]. We consider the distance D to some direct neighbor to be the number of transmissions necessary, on average, before one is received at that neighbor. The distance metric for all other nodes in the mesh network that are not neighbors is the sum of the distance advertised by the witness plus the distance to the witness. In order to calculate D we need to measure p, the probability of receiving a Presence Beacon from some neighbor. In some time

period T we expect to have $\frac{T}{\tau}$ arrivals from each neighbor, but we actually receive r. Therefore:

$$p = \frac{r}{\frac{T}{\tau}} = \frac{r \cdot \tau}{T}$$

We define D as:

$$D = log\left(\frac{1}{p}\right) = -log(p)$$

This definition allows us to add the distances of consecutive links along a path —which is intuitive from a human perspective—, while preserving the end-to-end probability of a successful transmission along the path in the summed distance value as being the product of all successful transmission probabilities along the path. We prove this as follows: Let p_i be the individual probabilities of successful transmission and D_i be the respective distances for each link i along some path. Let D be the total distance and p be the end-to-end probability of a successful transmission. Then:

$$D = \sum_i D_i \tag{3.1}$$

$$-log(p) = -\sum_i log(p_i) \qquad \text{(by definition of distance)} \tag{3.2}$$

$$log(p) = log\left(\prod_i (p_i)\right) \tag{3.3}$$

$$p = \prod_i (p_i) \tag{3.4}$$

and so we preserve the total probability by adding the respective distances.

3.2.5 Parsing Presence Beacons

For every entry in a Presence Beacon that node N received from some source node S, N first goes through a series of checks to validate the entry. If the entry is valid, N updates its Presence Table accordingly. N first checks that the reported target node in the entry is not N itself. If it is, then N stores the reported distance and discards the rest of the entry. The distance reported by S represents the uplink distance from N to S, as we show in

Section 3.2.4. Then, N checks that the reported witness is not N itself, as this is information that N already knows about. Finally, N checks the reported sequence number s against the sequence number s' that N already has stored about the target node of this entry; if $s > s'$ then this entry is more fresh and therefore it is processed, otherwise it is discarded. If this is the first entry received about this target node, the reported sequence number is stored and the entry is processed.

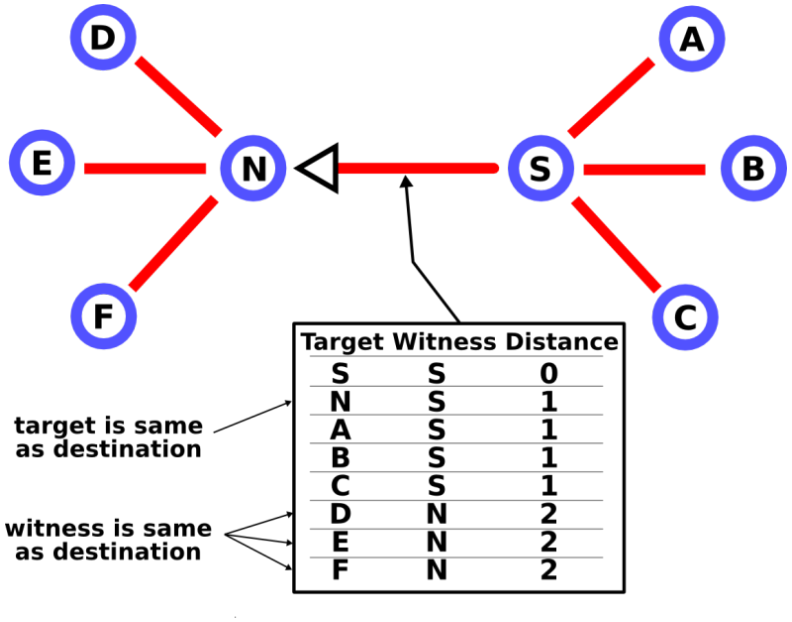


Figure 3-2: Contents of Presence Beacon sent from S to N: The second and the last-three presence entries are skipped at node N as they do not provide useful information.

Perkins et al. [66] proved that the use of sequence numbers breaks any circles that may be formed by presence entries that get forwarded in the network about a target that is no longer existent. A similar problem known as "Count to Infinity" also arises in distance-vector algorithms, such as Bellman-Ford. Sequence numbers address these problems in a deterministic and efficient way, as shown by Perkins.

The Presence Table maintains a list of all known nodes, the total distance and path to reach each node and a list of the times of all previous Presence Entry arrivals for each node using some specific path. There may exist multiple paths to reach a node and separate entries in the Presence Table represent each path.

All Presence Beacons received from neighbors are used to update the downlink distance [74] from each respective neighbor. The distance to each neighbor is calculated as shown in Section 3.2.4. We only calculate distances for direct neighbors and construct distances to remote nodes by adding the distance to the target node reported by the neighbor and the distance to the neighbor.

Valid presence entries are used to update the sequence number of the target node and the respective list of Presence Entry arrivals, which in turn is used to calculate the next arrival estimate.

3.2.6 Constructing new Presence Beacons

We construct a new Presence Beacon by following the format described in Section 3.2.3. For each target node we know about in our Presence Table, we construct a Presence Entry and use as witness the neighbor from which we received information about the target node. If there are multiple paths to the target, we use the one with the minimum distance. If we are constructing a Presence Entry about a neighbor, then both the target and witness are set as the neighbor itself. Section 3.2.11 demonstrates an extensive example of how Presence Beacons are constructed.

3.2.7 Density–based adaptation of period τ between transmissions

The traffic experienced by a node due to presence information received by its neighbors is dependent both the local density[2] and total size N of the network: A node will receive as many Presence Beacons per period as it has neighbors and each beacon size is proportional to the size of the network. As a result, if we do not adapt the rate at which neighbors are sending Presence Beacons and/or the number of nodes that each Presence Beacon covers, then the traffic overhead will grow as $O(N^2)$.

[2] *Local density* is equivalent to the number of neighbors K of a node.

As of this writing, Cerebro only adapts the rate at which neighbors transmit Presence Beacons. We ensure that the rate ρ of beacons received per second is kept constant by requiring the period τ at each neighbor to be:

$$\tau = \frac{K}{\rho}$$

For example, for power saving reasons we may require that we only receive 1 Presence Beacon every 3 seconds, irrespective of the number of neighbors K by having each neighbor adjust τ accordingly. Of course, there are cases where two neighbors have different degrees and therefore different values for K. This would result in an inconsistency in the respective values of τ, causing the neighbor with more (less) nodes to receive more (less) beacons than expected from the node with less (more) neighbors, because the latter belongs in a more sparse (dense) neighborhood. However, empirically, we have not encountered a realistic case in which the discrepancy in the degree of two neighbors is more than half an order of magnitude.

3.2.8 The Witness Field

The witness offers important information both for the operation of the presence protocol itself, but also for the applications that use Cerebro. The witness is the neighbor from which we receive presence information about some remote node. In each Presence Entry, the neighbor will report its own witness/neighbor from which it received information about some remote node. As a result, we can reconstruct the exact network layout up to two hops away along the path to some remote node. For example, in Figure 3-4, node A receives a Presence Entry from node B having node E as the target and node C as the witness. Therefore, node A can actually build a tree for the exact layout between A and C, but A cannot tell if E is a direct neighbor to C or not.

The witness field can be considered as the "next hop" in order to reach a remote destination. In the literature of distance-vector routing protocols, as in Cerebro's case, the next hop is inferred to be the neighbor that reports the shortest distance[3] to some remote node. However, to the best of our

[3] Different metrics for "distance" may be used, but the idea of the "next hop" is the same.

knowledge, the next hop itself is not reported in routing advertisements of known distance-vector protocols, therefore nodes cannot tell where a neighbor received a routing advertisement from. This has side effects like bouncing stale information between neighbors. Including the witness field in presence entries allows us to mitigate this problem, although sequence numbers are still necessary in order to break large cycles that involve more than three nodes.

Most importantly, the witness field allows us to distinguish where presence information is flowing from and in which direction. Looking again at Figure 3-4, node C can efficiently deduce that it is at a crossroad for presence information flowing from witnesses B, D and J which involves 2, 6 and 579 nodes respectively. Such information is fundamental for applications like locating major events in an urban environment. Similar applications that use Cerebro are discussed in more depth in Chapter 5. Although it is possible to achieve the same result by comparing sequence numbers received by neighbors, this would impose quadratic complexity.

3.2.9 Node State Inference

As the time we wait for a presence update to arrive from some node may increase infinitely, we need to apply a cutoff value to that waiting period beyond which we consider the node to be disconnected from the network. Instead of using some static value, we model the arrival of presence entries about each target node in the mesh network as a different Poisson process. We assume that each arrival of an entry about some node is independent from any previous entry arrivals for that node, which allows us to model entry arrivals as a Poisson process. Let λ be the rate of Presence Entry arrivals from some node S. In order to ensure a confidence level of p (eg. 99%) that the node is offline, the cutoff value for the waiting time t will be given by the cumulative distribution function (CDF) of the corresponding exponential distribution:

$$p = Pr(x \leq t) = 1 - e^{-\lambda t} \Rightarrow t = -\frac{log(1-p)}{\lambda} \tag{3.5}$$

For example, for $\lambda = 1$ arrival per second and $p = 0.99$, then $t \simeq 4.6$ seconds, which is the time we have to wait until it is 99% sure that the node is offline.

Because the arrival rate may change over time as nodes move and link quality fluctuates, we need to adapt the period T used to measure a neighbor's distance and arrival rate λ. If we choose T to be too large then our state inference will be accurate but not responsive (similar to choosing a low value for α in a low-pass filter). If we choose T to be too small, then our state inference will be responsive, but may produce false events such as a node appearing to be disconnected.

During period T_i we counted r_i Presence Entry arrivals for node i, which yields an arrival rate $\lambda_i = \frac{r_i}{T_i}$. In order to keep having sufficiently accurate measurements, we adapt period T_i', as:

$$T_i' = \frac{A}{\lambda_i} = \frac{A \cdot T_i}{r_i}$$

where A is the number of arrivals we expect to have during T_i' at the rate we measured during at T_i. We observed that setting $A = 10$ is sufficiently large to produce an accurate measurement, while maintaining responsiveness.

Figure 3-3 shows an example of arrivals from some remote node during two consecutive time periods T and T'. The first one is $T = 10$ secs and we expect $A = 10$ arrivals, but only receive 7. We therefore increase $T' = 14$.

3.2.10 Algorithm Properties and Theoretical Analysis

We assess the scaling properties of our presence protocol as a function of the number of nodes N that participate in the network. We consider a dense scenario where all nodes are in range with each other forming a full mesh network and a sparse scenario in which nodes have only two neighbors, but form a long, multi-hop network. In each case we analyze the respective

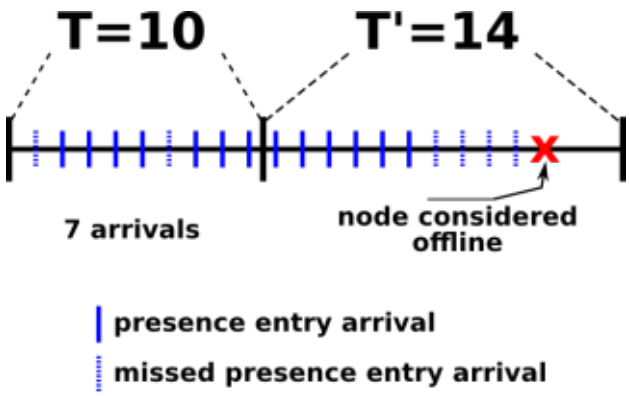


Figure 3-3: Presence entry arrivals from some remote node during two consecutive periods T and T'. Two arrivals were missed during the first period and the second period was adjusted accordingly. However, before the end of the second period, we lost more than 4 consecutive arrivals and the node is considered disconnected.

network overhead which is the aggregate amount of traffic imposed by the algorithm, algorithm complexity which hints the CPU cycles necessary to run the algorithm and responsiveness as indicated by the minimum time required to get notified about a remote node joining/leaving the network.

Traffic overhead

Theorem: The total traffic overhead in some area of the network where at most K nodes are in range with each other and the network contains a total of N nodes, grows as $O(N)$.

Proof: Each Presence Beacon contains N presence entries, each Presence Entry is 15 bytes long, for a total of $15 \cdot N$ bytes/beacon. When all K nodes discover each other, they adapt their sending rate by setting $\tau = \frac{K}{\rho}$ (see also Section 3.2.7). Each node contributes to the network one Presence Beacon every τ seconds, therefore the total traffic overhead is:

$$O_{traffic} = \frac{15 \cdot N^2}{\tau} = 15 \cdot N \cdot \rho = O(N)$$

In the future, we will also be able to limit the parts of the network for which we provide presence information based on different criteria, such as distance and the social network of the user, as presented in Section 7.1, thus further

decreasing the traffic overhead, while making available information more relevant to the user.

Processing power overhead

CPU usage is affected both by the density of the network around the node and by the total number of nodes in the network. We assess processing overhead by examining the average number of presence entries that need to be processed every second, as a function of the total number of nodes and neighbors.

Based on Section 3.2.7, a node that belongs in a network with N nodes in total, of which K are neighbors, will receive as many as K Presence Beacons per period τ. In every beacon, a neighbor provides $O(N)$ presence entries, for a total processing overhead of:

$$O_{cpu} = \frac{K \cdot N}{\tau} = N \cdot \rho = O(N)$$

Responsiveness

Responsiveness is another characteristic that we need to pay attention to, especially for networks that span hundreds of nodes. In this section, we are interested only in the time overhead introduced by the algorithm itself, not in the overhead due to retransmissions, OS queuing and processing. We therefore treat the links to be perfect and OS delays to be negligible. However, we should acknowledge that if a Presence Beacon containing information about some node joining the network were lost, it would not be retransmitted until the next period τ, thus the delay between "retransmissions" is actually τ. Still, actual deployments [56], [84] have shown that blindly retransmitting (also known as "flooding") presence information as triggered by events can result in a broadcast storm that render the whole network unusable. As a result, we do not consider flooding to be a viable option.

Again, for a node that belongs in a network with N nodes in total of which K are neighbors, we measure the time necessary for the node to receive

information about an event at the edge of the network. This event can be a node joining or leaving the network. We are only interested in node arrivals and departures because topology changes that happen more than two hops away from some node is information that is lost before reaching the node.

The time necessary for an event to propagate throughout the network in the form of Presence Beacons that are sent along some path[4] is the sum of the individual periods τ_i for each edge i along the diameter of length D. There are $D-1$ edges along this path, therefore the propagation time overhead is:

$$O_{propagation} = \sum_{i}^{D-1} \tau_i$$

where, $\tau_i = \frac{K_i}{\rho}$.

In the special case where all nodes are layed down on a straight line, $K_i = 2, \forall i \in D$, so:

$$O_{propagation} = \sum_{i}^{D-1} \tau_i = \sum_{i}^{N-1} \frac{2}{\rho} = \frac{2(N-1)}{2} = \Omega(N)$$

At the other extreme, if the network is fully connected and the nodes are uniformly layed out on the plane, then the maximum length of the diameter is $D = O(\sqrt{N})$. This is straightforward to show in the case of a circle ($D = 2\sqrt{\frac{N}{\pi}}$) or square ($D = \sqrt{N}$). As this is a fully connected network, $\tau_i = \frac{N}{\rho}$. Then the propagation overhead becomes:

$$O_{propagation} = \sum_{i}^{D-1} \tau_i = \sum_{i}^{N-1} \frac{2N}{\rho} = O(N\sqrt{N})$$

However, this is an overestimation of the overhead because it assumes that as the event propagates from one side of the diameter to the other, it will never reach the target destination directly, but instead go through every single node along the diameter, although this is a fully connected network. In reality, even if the probability of a successful reception at the target destination is as low as $p = 0.5$, there is less than 10% probability that the event will not be received after 4 consecutive transmissions ($p^4 = 0.0625$),

[4] In the worse-case, this path is the diameter of the network.

independently of the size N. Thus, although the straight line layout may seem to scale better than a fully connected network in terms of propagation overhead, in practice the former is the worse case scenario.

We have shown that Cerebro's design provides a trade-off between responsiveness and traffic/processing overhead. At the expense of $\Omega(N)$ responsiveness in the worse case, we achieve linear traffic and processing cost with the number of nodes in the network.

3.2.11 An Extensive Example of the Presence Algorithm

In Figure 3-4, ten nodes form a mesh network. The solid lines represent high quality links, while the dashed ones represent lower-quality links. Each number next to a link is the distance between the nodes, which we assume to be the same in both directions.

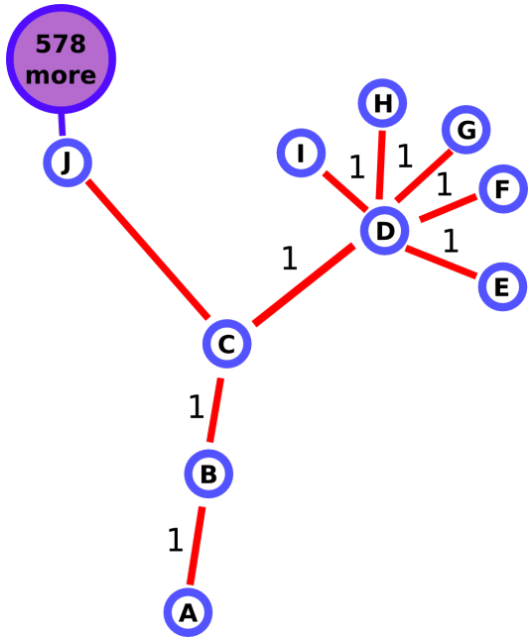


Figure 3-4: Example network layout

We assume that only nodes connected with solid lines can communicate with each other and the number next to each line shows the distance of the corresponding nodes. The following tables indicate on the left side the contents of the Presence Beacons as broadcasted by different nodes, while on the right side the resulting Presence Table is displayed as formed at the end of period τ.

time	**Beacon that each node will send**				**Presence table at end of period τ**			
	Sender	Target	Witness	Distance	at node	Target	Path	Distance
$t=0$	A	A	A	0	A:	B	B	1
	B	B	B	0	B:	A	A	1
						C	C	1
	C	C	C	0	C:	B	B	1
						D	D	1
	D	D	D	0	D:	C	C	1
						E	E	1
						F	F	1
	E	E	E	0	E:	D	D	1
	F	F	F	0	F:	D	D	1
	Sender	Target	Witness	Distance	at node	Target	Path	Distance
$t=1$	A	A	A	0	A:	B	B	1
		B	B	1		C	B,C	2
	B	B	B	0	B:	A	A	1
		A	A	1		C	C	1
		C	C	1		D	C,D	2
	C	C	C	0	C:	B	B	1
		B	B	1		D	D	1
		D	D	1		E	D,E	2
						F	D,F	2
						A	B,A	2
	D	D	D	0	D:	C	C	1
		C	C	1		E	E	1
		E	E	1		F	F	1
		F	F	1		B	C,B	2
	E	E	E	0	E:	D	D	1
		D	D	1		F	D,F	2
						C	D,C	2
	F	F	F	0	F:	D	D	1
		D	D	1		E	D,E	2
						C	D,C	2

Table 3.2: Presence Tables at times 0 and 1: Initially all Presence Tables are empty and each node starts by sending a Presence Beacon at the end of the first period.

Initially, no node receives any beacon from their neighbors and they all start by emitting a beacon about themselves. At the end of the period, each node has information about its neighbors. This information is placed into a new beacon which is sent during the next period. As protocol operation continues, information about node E propagates along the path down to node A and will take four periods[5] to reach A.

Let us look into the information that node A receives during the fourth period ($t = 3$) in more detail. At the end of the third period, node B will

[5] We assume distances to be 1, which implies perfect links.

time	**Beacon that each node will send**				**Presence Table at end of period τ**			
	Sender	Target	Witness	Distance	at node	Target	Path	Distance
	A	A	A	0	A:	B	B	1
		B	B	1		C	B,C	2
		C	B	2		D	B,C,D	2
	B	B	B	0	B:	A	A	1
		A	A	1		C	C	1
		C	C	1		D	C,D	2
		D	C	2		E	C,D,E	3
						F	C,D,F	3
	C	C	C	0	C:	B	B	1
		B	B	1		D	D	1
		D	D	1		A	B,A	2
		E	D	2		E	D,E	2
		F	D	2		F	D,F	2
$t = 2$		A	B	2				
	D	D	D	0	D:	C	C	1
		C	C	1		B	C,B	2
		E	E	1		A	C,B,A	3
		F	F	1		E	E	1
		B	C	2		F	F	1
	E	E	E	0	E:	D	D	1
		D	D	1		F	D,F	2
		F	D	2		C	D,C	2
		C	D	2		B	D,C,B	3
	F	F	F	0	F:	D	D	1
		D	D	1		E	D,E	2
		E	D	2		C	D,C	2
		C	D	2		B	D,C,B	3

Table 3.3: Presence Table time $t = 1$: All nodes now have information about all other nodes, except node A that still has not received information about nodes E and F and vice versa for nodes E and F.

have information about all the nodes and their actual respective paths from B. In the fourth period, specifically for nodes E and F, node B be will report to A that C is the witness for nodes E, F and therefore information about node D being in between C and E, F will be lost. The resulting Presence Table at node A is shown in Table 3.4.

time	**Beacon that each node will send**				**Presence Table at end of period τ**			
	Sender	Target	Witness	Distance	at node	Target	Path	Distance
$t = 3$	A	A	A	0	A:	B	B	1
		B	B	1		C	B,C	2
		C	B	2		D	B,C,D	3
		D	B	2		E	B,C,E	4
						F	B,C,F	4
	B	B	B	0	B:	A	A	1
		A	A	1		C	C	1
		C	C	1		D	C,D	2
		D	C	2		E	C,D,E	3
		E	C	3		F	C,D,F	3
		F	C	3				
	C	C	C	0	C:	B	B	1
		B	B	1		D	D	1
		D	D	1		A	B,A	2
		E	D	2		E	D,E	2
		F	D	2		F	D,F	2
		A	B	2				
	D	D	D	0	D:	C	C	1
		C	C	1		B	C,B	2
		E	E	1		A	C,B,A	3
		F	F	1		E	E	1
		B	C	2		F	F	1
		A	C	2				
	E	E	E	0	E:	D	D	1
		D	D	1		F	D,F	2
		F	D	2		C	D,C	2
		C	D	2		B	D,C,B	3
		B	D	3		A	D,C,A	4
	F	F	F	0	F:	D	D	1
		D	D	1		E	D,E	2
		E	D	2		C	D,C	2
		C	D	2		B	D,C,B	3
		B	D	3		A	D,C,A	4

Table 3.4: Presence Tables at time $t = 3$: At this point all Presence Tables stabilize and do not change, unless there is change in the topology of the network.

3.3 The Collaboration Mechanism

Having established a Presence Table, each node has information about the unique ID and the distance to other nodes in the network. However, additional mechanisms are necessary before each node can run any applications that make use of presence information, as discussed in Section 3.2.1. First, each node needs to share more than just its unique ID. We need to represent presence information in a compact and extensible way. Second, a

mechanism for sharing presence information as efficiently as possible is necessary. We need to account for potentially high mobility and low available bandwidth.

3.3.1 Node Profiles

A node profile contains presence information about some node in the network, as described earlier in Table 3.1. This information represents the user and the node as a whole. We need to account for a variety of applications, therefore a profile may contain information ranging from the user's name and picture, to a list of services and applications that the node can run collaboratively with other nodes in the network.

We employ a simple list of recursive key/value pairs to represent node profiles. A key/value pair can represent a simple variable, such the node's nickname, or contain a list of key/value pairs of its own. For example, Figure 3-5a shows the profile of a node that is operated by a user that bares two profiles, is running two instances of an application, offers a service and has 3 types of radios. More specifically, the first identity reveals some biographical information about the user, the device's geographical coordinates, a list of applications that the node is able to run collaboratively with other nodes and a unique ID to break ties with other profiles that the node may have.

Figure 3-5b shows the profile of a node that one would typically find attached to a wall in some common area. This node allows other nodes in the network to localize themselves by sharing its geographical coordinates. Additionally, it advertises that it shares an internet connection with up to 20 users at a speed up to 500Kbps. Finally, it offers a total of 500GB of storage capacity and can accept processing jobs to offload CPU usage from mobile nodes. No more information is offered about each service, so the node can be queried for more information, if available.

3.3.2 Teleporter: Reliable, Parallel Data Transport

Having established a scalable protocol that provides a list of all nodes in the network and an extensible structure for presence information, we need a mechanism to exchange this information. *Teleporter* is the module responsible of transferring data between neighbors over some wireless medium.

Cerebro is designed to be used primarily over a wireless medium and Teleporter is exploiting this medium's fundamental characteristic — its broadcast nature. A unicast transmission[6], on which TCP is based, over a wireless medium is essentially an abstraction of a broadcast which is discarded on all neighbors but the stated receiver. However, this scheme in a wireless setting is largely wasteful, because the same data will be transmitted as many times as there are receivers. In an environment where new nodes are constantly encountered and our profile and data files need to be shared with multiple neighbors, establishing a TCP/IP session per recipient is inefficient.

Traditionally, the solution has been to limit the number of recipients as much as possible. Teleporter turns the broadcast nature of the wireless medium into a feature by allowing all neighbors to opportunistically receive a broadcast transmission, while data is being reliably transferred to a single neighbor. As we show in Section 6.1, it is possible to use this technique to transfer large chunks of data to multiple neighbors, while sending data reliably to a single destination at a time. After having transferred all data to the first neighbor, we only need to transfer the missing pieces to the rest of the neighbors, rather than all data from scratch, as shown in Figure 3-6.

Teleporter, as of this writing, does not offer a congestion-control mechanism. However, congestion cannot form between two neighbors because there are no intermediate nodes on which traffic is temporarily stored with the potential to create congestion. Teleporter employs NACKs[7] that notify the sender of missing pieces of data at the receiver. If a receiver happens to be in the middle of multiple concurrent receptions and one of them fails, the

[6] In a unicast transmission there is a single source and a single destination
[7] NACK stands for Not-ACK, a negative acknowledgment.

sender will implicitly become aware of this by the absence of a NACK that it is expecting.

As long as the sender keeps getting NACKs back from the receiver, the sender will keep sending pending pieces of data. This data transport mechanism between neighbors converges either to eventual successful reception of all data, or failure is reported. A failure is reported if we no longer receive NACKs from the destination. This may be because the node is no longer in range, or data and/or NACKs are lost because of traffic overload, or the destination is no longer interested in the rest of the data (for example, reception was canceled by the user).

On the sender's side the protocol goes as follows: In the setup phase, the sender counts the total number of frames[8] T to be sent and assigns a number to each one in the range $(0, T-1)$.

Next, the sender sends a burst of data that contains B frames, or less if $sizeof(N) + sizeof(P) < B$. The first B frames are chosen from the set P of pending frames. If P contains less than B frames, $B - sizeof(P)$ more frames are chosen from N. Chosen frames, are only marked for transmission, but not removed from the sets yet.

All B frames are sent at maximum speed, with no delay in-between each frame. At the end of the burst, the sender expects to receive a NACK and will keep sending up to k more frames from P — or N, if P is empty — every d msecs. If no NACK is received at the end of the additional k frames that were sent on top of the burst size B, then a failure is reported.

If a NACK is received, the frames that were sent from sets N and/or P are removed and all reported missing frames are added to pending set P. The NACK also contains the greatest index number i of all frames received so far. The sender also maintains a pointer j to the greatest index transmitted so far and will place all frames in between $(i+1, j)$ into P.

The process is repeated until both sets N and P are empty or a failure is reported.

[8] Frames are the equivalent of packets at Layer-2.

On the destination's side the protocol is as follows: Each frame received states its index number, the total number of frames involved in this transmission, the burst size B and whether the frame was destined to the current node. If the node is the destination of this frame, the frame is processed, otherwise it is cached. If it is set to be processed, the total number of frames r received as part of the current data transmission is compared against the reported size B. If $r \geq B$ then a NACK is constructed containing all missing frame index numbers, up to the greatest index number received so far and r is reset to 0. In other words, since frame index numbers "in the future" cannot be reported in the NACK, index i informs the sender of non-received pieces following the greatest index received.

Variable	Description
T	Total number of frames to be sent
N	Set of new frames that have not been transmitted before
P	Set of pending frames that were not received, as reported by NACKs
B	Burst size in number of frames
r	Number of frames received at the destination so far
k	Number of additional frames to be sent at the end of a burst
d	Delay introduced between consecutive transmissions, while expecting a NACK
i	Pointer to the greatest frame index number received so far
j	Pointer to the greatest frame index number sent so far

Table 3.5: Summary of variables employed in Teleporter.

Table 3.5 summarizes the variables involved in Teleporter's transmission protocol, both for the sender and the receiver.

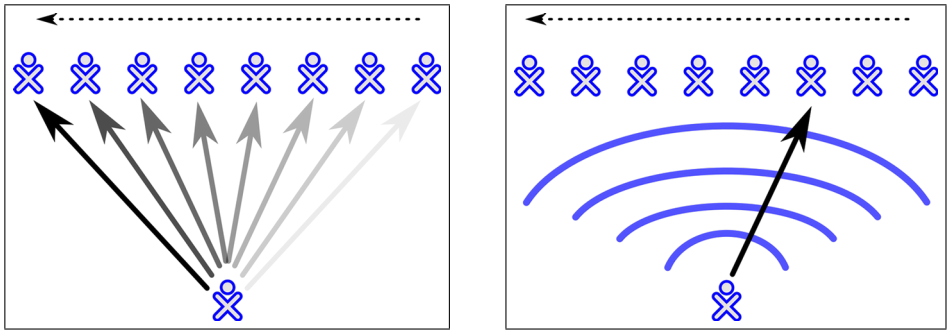
```
{
    "profile1" : {
        "name"      : "John",
        "surname"   : "Doe"
        "profession": "MD"
        "LAT"       : "39.9789"
        "LON"       : "22.6280"
        "apps"      : "chat"
        "uniqueID"  : "ID1"
    },
    "profile2" : {
        "name"     : "Phalse",
        "in1"      : "MP3 share"
        "file1"    : "song1.mp3"
        "file2"    : "video1.avi"
        "apps"     : "file-sharing"
        "uniqueID" : "ID2"
    },
    "application1" : {
        "name" : "chat"
        "room" : "Politics"
    }
    "application2" : {
        "name" : "chat"
        "room" : "Saturday Night"
    }
    "service1" : {
        "type"  : "internet"
        "limit" : "5 users"
        "speed" : "20kbps"
    }
    "radios" : {
        "radio1" : "WiFi"
        "radio2" : "Bluetooth"
        "radio3" : "GPS"
    }
}
```

(a) The user has two profiles and may choose to expose any one of them at a time. The node is running two instances of the Chat application, it offers an internet sharing service and has 3 types of radios.

```
{
    "profile1" : {
        "name" : "Host-153",
        "LAT"  : "39.9789"
        "LON"  : "22.6280"
        "uniqueID"  : "ID1"
    },
    "service1" : {
        "type"  : "internet"
        "limit" : "20 users"
        "speed" : "500Kbps"
    }
    "service2" : {
        "type"     : "storage"
        "capacity" : "500GB"
    }
    "service3" : {
        "type"  : "processing"
        "speed" : "4GHz"
    }
    "radios" : {
        "radio1" : "WiFi"
        "radio2" : "Bluetooth"
        "radio3" : "GPS"
    }
}
```

(b) This is an example of a fixed node that offers geographical coordinates, an internet connection, 500GB of storage capacity and can accept processing jobs to offload CPU usage from mobile nodes.

Figure 3-5: Node profile examples of a mobile, attended node and a static, non-attended one.



(a) Sequential data transfer using TCP from rightmost to leftmost neighbor.

(b) Parallel data transfer using Teleporter: data is reliably transferred to one destination at a time, while other nodes opportunistically cache all received data.

Figure 3-6: TCP vs. Teleporter for data transfer to multiple neighbors

Chapter 4

System Implementation

In this chapter we present Cerebro's implementation by looking at its core modules in more detail and describe the application programming interface (API) that allows developers to build on top of Cerebro.

4.1 Cerebro Modules

The first prototypes of Teleporter (see also Section 3.1) were functional in that they allowed discovery of devices and file reconciliation of files stored on them, but the tight coupling of the discovery mechanism and the reliable file transfer made Teleporter extremely hard to debug and to extend its functionality. Building on lessons learned from the previous implementations of Teleporter, Cerebro now comprises four basic modules, each one in charge of a well-defined task. The resulting modules are shown in Figure 4-1.

Cerebro is built using Python and each module shown in Figure 4-1 is a python module. Other modules that are not shown in the Figure include:

node_struct Implements the data structure used to represent nodes in the network tree.

common Contains environment variables that control Cerebro's behavior

demjson Implements serialization/de-serialization of byte streams to and from JSON [26] strings.

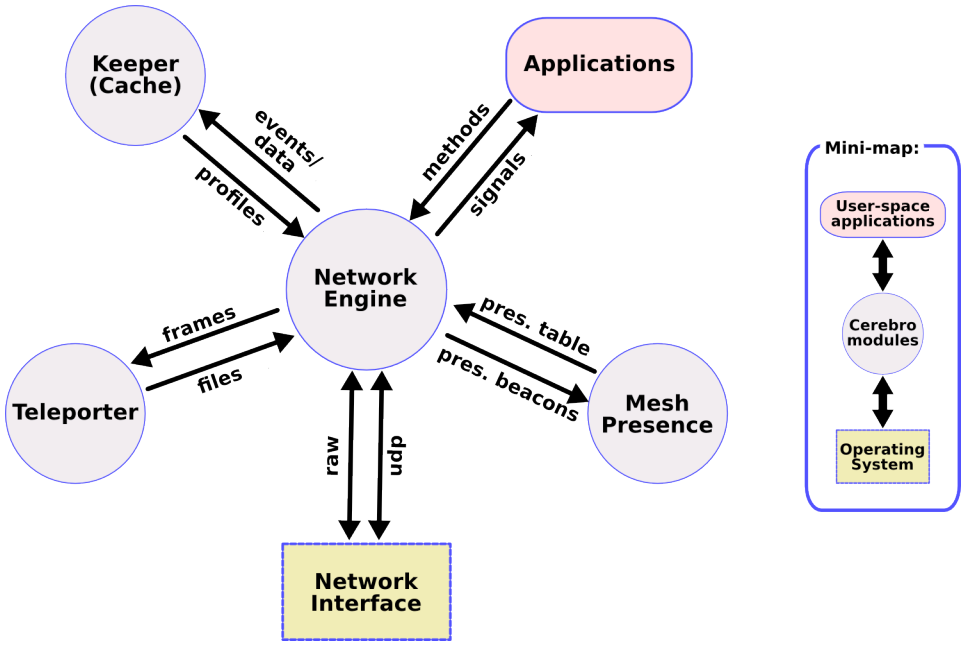


Figure 4-1: Cerebro architecture

Before introducing each module individually, we should underline the two basic software design decisions that are driving this architecture:

1. **Modules do not communicate directly**, except through the *network_engine*
2. **Only the *network_engine* communicates with the network interface directly**

As a result:

- debugging of each individual module is made simpler because interaction with other modules is minimized and well-defined.
- it is possible to interchange existing modules with others without affecting the rest of the architecture, eg. replace *Teleporter* with an IP-based solution.

- network security is improved because there is central control (ie. the *network_engine*) over all ingress/egress traffic. As a result, it is straightforward to introduce rules similar to a network firewall.

discuss node entering network

4.1.1 Mesh Presence

Presence information is the epicenter of Cerebro's functionality and providing it is the objective of the *mesh_presence* module. Based on presence beacons that this module receives from the *network_engine*, it builds a network tree rooted at the current node. An example of such a tree is shown in Figure 3-4. If there are multiple paths to a destination, then each path is treated as a separate branch in the tree. The data structure of each node in this tree is represented by module *node_struct*. The latter provides a *Node class* for easy manipulation of tree nodes such as adding and removing branches, adding presence entry arrivals, pre-order and in-order tree traversal, etc.

mesh_presence processes each presence beacon (*process_frame*) by breaking down into its presence entries.and then processing each one (*process_presence*) independently. Each presence entry is processed as presented in Section 3.2.5. Each valid presence entry either creates a new node in the network tree, or adds an arrival for some existing node in the network tree.

Periodically, the network tree must be maintained and a new presence beacon must be formed (*do_maintenance*) to be broadcasted. Maintenance involves removing branches of the tree that are rooted at some node for which we have not received a presence entry in some time, as described in Section 3.2.9 for inferring node state. When maintaining the network tree we keep all branches, even when we have multiple paths of varying distances to the same destination. This is because it takes several presence entry arrivals to obtain an accurate distance estimate and we want to be able to switch to an alternative path instantly, once the best path we have breaks or degrades. As part of the maintenance, we also build the minimum spanning tree of the network by obtaining the minimum distances to each node (*build_mindists*).

In the last step, we formulate a new presence beacon from the minimum distances to each node and hand over the beacon to the *network_engine* to be broadcasted. Figure 4-3 shows the payload of an example presence beacon.

Finally, the *mesh_presence* module keeps a per-node history of presence entry arrivals, thus allowing applications to obtain information about presence patterns in the past.

The *mesh_presence* module can be used to answer simple but useful queries such as:

1. Give me a list of all neighbors.
2. Is node X present in the network and if yes, what is the distance to it?
3. When was the last time we heard from node Y?
4. What is the mean presence entry arrival time and variance for node Z?

By combining the above queries with presence history information we can obtain co-presence patterns of different nodes, associate a neighbor's presence with the physical presence of some user, etc.

4.1.2 Teleporter

The *teleporter* module provides reliable one-to-many data transport. It uses the network sockets provided by the *network_engine* module and provides a data transport service to one or more destinations. When a data reception is completed, ie all pieces for some data transmission are received, *teleporter* notifies *network_engine* accordingly by means of a callback function. All interaction between *teleporter* and *network_engine* is summarized in Table 4.1.

As of this writing, *teleporter* uses three types of frames, as shown in Table 4.2. The `REQ`frame type is phased for removal from *teleporter*, as it is redundant. *teleporter* should only provide reliable data transfer and data requests are application-specific and should be treated by *teleporter* simply

Method	Direction	Description
`process_data`	IN	Processes all frames of type ACK, DAT and REQ.
`send_request`	IN	Sends a request to some destination
`send_multicast_request`	IN	Performs sequential `send_request`calls to multiple destinations
`push_data`	IN	Initiates a data transmission session to some destination.
`push_multicast_data`	IN	Performs sequential `push_data`calls to multiple destinations
`handle_new_data`	OUT	Callback to pass data to *network_engine* when a data reception completes
`handle_new_data_from_cache`	OUT	Callback to pass data to *network_engine* from the cache, if it already exists
`in_cache`	OUT	Checks whether data already exists in cache
`handle_new_req`	OUT	Callback to pass a received request to *network_engine* to be processed
`send_fracket`	OUT	Provides *teleporter* access to *network_engine*'s raw and UDP sockets to send a frame and/or packet respectively

Table 4.1: Methods used for all interaction between *teleporter* and *network_engine*. Methods with direction marked as IN are defined in *teleporter* and called from *network_engine* and vice versa.

as "data" fitted in `DAT`frames. Figure 4-4 summarizes the headers used by *teleporter* for the different frame types that *teleporter* employs.

Frame type	Description	Destination
`DAT`	Data frame	Pseudo-Unicast
`ACK`	Acknowledgment frame	Unicast
`REQ`	Request frame	Unicast

Table 4.2: Frame types used by *teleporter*.

Data frame types are sent to the broadcast address in order to reach all neighbors. However, a destination address is inserted into the payload, thus notifying the intended destination to respond by acknowledging reception, while other neighbors passively cache received data. We call this technique "pseudo-unicast" since it is the essentially the opposite to "pseudo-broadcast", in which a frame is sent to some destination over unicast, but other nodes in promiscuous mode receive the transmission as well. The differences between the two techniques are summarized in Table 4.3.

	pseudo-broadcast	pseudo-unicast
TX speed	Unicast rate	Broadcast rate
privileges required	elevated (root)	default user
promiscuity required	yes	no
layer-2 acks	yes	no
layer-3 acks	yes	yes

Table 4.3: Comparison of "pseudo-broadcast" vs. "pseudo-unicast".

While the unicast rate is generally greater than the broadcast rate —which is usually fixed at 1Mbit— some network interfaces[1] offer the ability to set the broadcast rate higher than the default. The driving characteristic that makes the use of pseudo-broadcast prohibitive is the promiscuity requirement; placing the network interface in promiscuous mode increases the amount of processing power required to process all captured frames and voids any power-saving scheme that would place the device's processor into sleep mode.

4.1.3 Network Engine

The *Network Engine* is the module that "glues" all Cerebro modules together. It also provides control over the network sockets used throughout Cerebro and exposes an API to develop applications that use Cerebro as a service.

Cerebro employs raw sockets in order to make direct use of layer-2 frames. This is primarily because all nodes in the network need to be physically identified by a unique ID and the layer-2 (MAC) address of the wireless interface of each node seemed to be the best candidate to be used as per-node unique ID. Also, using raw sockets eliminates the need to establish unique IP addresses in a mobile network. However, as we establish in the Section for future work (7.1), a node's identity should not be verified by means of its unique ID, because the latter should be used as a means for communication, not identification.

The *Network Engine* uses two raw sockets and a UDP socket, as described in Table 4.4.

[1] Marvel's 88W8388 chip allows setting the broadcast rate up to 54Mbps.

Socket name	type	Description
`DATsock`	raw	Used for *teleporter* frames of types `DAT` and `ACK`.
`PRESsock`	raw	Used for *mesh_presence* frames.
`udpsock`	UDP	Used to wrap raw frames in UDP packets.

Table 4.4: Sockets used by *Network Engine*.

We use separate raw sockets for *mesh_presence* and *teleporter* modules to avoid having to check whether each incoming frame should be forwarded to one module or the other. That would involve a two-byte string comparison for every incoming frame and this could impose a slight delay on weak CPUs for data streams arriving at 1000 packets/second.

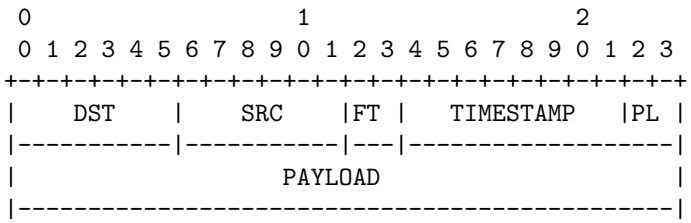
We also use a UDP socket to encapsulate Cerebro's raw frames into UDP packets. This way we can connect remote mesh networks over the internet by means of a node that acts as internet gateway in each mesh network. Furthermore, some operating systems do not support the use of raw sockets, thus using UDP encapsulation we overcome this problem without any changes necessary in the existing code that builds outgoing and parses incoming frames. The source IP address used with UDP packets can be based on the MAC address of the interface, as shown in appendix A.2, to ensure uniqueness without the need for an address conflict resolution protocol.

Figure 4-2 shows the frame header used in *Network Engine* for all data exchanged in Cerebro. The `PAYLOAD` is specific to each module and the two types of payload are shown in Figures 4-3 and 4-4.

4.1.4 Keeper

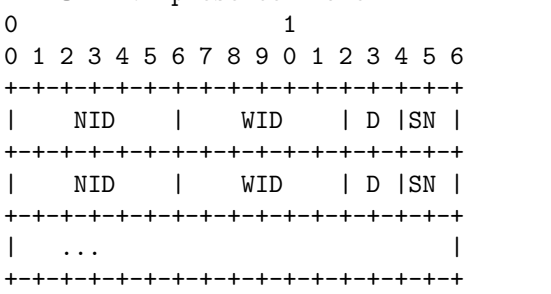
The *keeper* module maintains the node profile of the current node and that of other nodes for which we have received their profile. A python dictionary is used to store profiles and *keeper* serializes a profile for transmission (`pack_status_info`) and de-serializes a received profile into a python dictionary (`unpack_status_info`).

keeper also provides class SAS (Shared Application Structure), which holds information about the applications that each known node, including the current node, is currently sharing or is willing to share with others in the




```
DST      : destination ID  (6 bytes)
SRC      : source ID       (6 bytes)
FT       : frame type      (2 bytes)
TIMESTAMP: frame timestamp (8 bytes)
PL       : payload length  (2 bytes)
PAYLOAD  : payload (1 or more bytes)
```

Figure 4-2: Cerebro's frame header. The DST and SRC fields represent network interface MAC addresses. The contents of PAYLOAD can be either a Teleporter or a *mesh_presence* frame, as described in Figures 4-3 and 4-4 respectively.




```
NID : target node ID  (6 bytes)
WID : witness node ID (6 bytes)
D   : distance        (2 bytes)
SN  : sequence number (2 bytes)
(1 or more repetitions of all 4 fields)
```

Figure 4-3: Presence frame header

network. SAS can be queried for information on specific nodes or applications.

4.2 Application Programming Interface (API)

Cerebro is meant to be used as a service by other applications, therefore it offers a comprehensive API that includes methods to control Cerebro's

```
PAYLOAD for generic teleporter frame

0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T|BS |PRT|PL |    TELEPORTER PAYLOAD   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

T  : teleporter frame type (1 byte)
BS : burst size            (2 bytes)
PRT: port                  (2 bytes)
PL : payload length        (2 bytes)
```

Figure 4-4: Teleporter frame header. The contents of TELEPORTER PAYLOAD can be any of the two described in Figure 4-5.

```
TELEPORTER PAYLOAD for ACK frame
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                FHASH                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|LA |NA |NA |NA |NA |...                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

LA : last piece acked      (2 bytes)
NA : NACKed piece          (2 bytes)
(0 or more repetitions of the NA field)
```

```
TELEPORTER PAYLOAD for DAT frame
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                FHASH                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    DST    |PN |TP |     DATA...         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

DST  : destination      (6 bytes)
PN   : piece number     (2 bytes)
TP   : total pieces     (2 bytes)
DATA : data to be sent  (1 or more bytes)
```

Figure 4-5: Frame headers for ACK and DAT teleporter frames

behavior and to use its services and signals to which an application can subscribe to be notified of various events. The methods and signals that Cerebro exposes are implemented using D-Bus [33], a well-known message bus system for inter-process communication (IPC). D-Bus is available on all major Linux distributions, including the XO laptop by OLPC [55] and the OpenMoko Freerunner mobile phone [58].

Appendix A.1 provides a detailed description of the API that includes the input and output of each signal and method.

Chapter 5

System Applications

This chapter discusses about applications that can be built on top of Cerebro and the way they differentiate themselves from previous work done in their respective application areas. First, we present three application designs involving discovery of humans and objects in the same physical area, while providing file sharing, chat and a bidding game between users. Then, we present a list of applications that could further benefit from Cerebro's features to provide discovery, context-awareness, communication and group establishment for devices in the same physical area.

5.1 Discovery of Humans and Objects

A fundamental application for Cerebro has been the discovery of humans and objects in physical proximity. The first attempt to display other devices in physical proximity was "Space", a Sugar [14] activity for OLPC. "Space" is described in more detail in Section 5.1.3.

5.1.1 Web-based UI

The second attempt was based on a web interface showing other devices layed out in a two-dimensional space, as shown in Figure 5-1. In a previous version, Cerebro provided a basic web server implemented in Python that

served an AJAX-based web page that would periodically poll the Cerebro service for updates in the network layout. Cerebro would pass to the browser a list of pairwise distances between nodes, and the web browser would use a spring-force placement algorithm [67] implemented in Javascript[1] to come up with the right coordinates on which the nodes would be placed. However, the placement was only accurate for direct neighbors.

The web-based interface was discontinued because continuously polling the web server for changes was inefficient because it required significant processing power to recalculate the network layout and required a constant flow of data between the Cerebro service and the web UI, even when there were no changes in the layout.

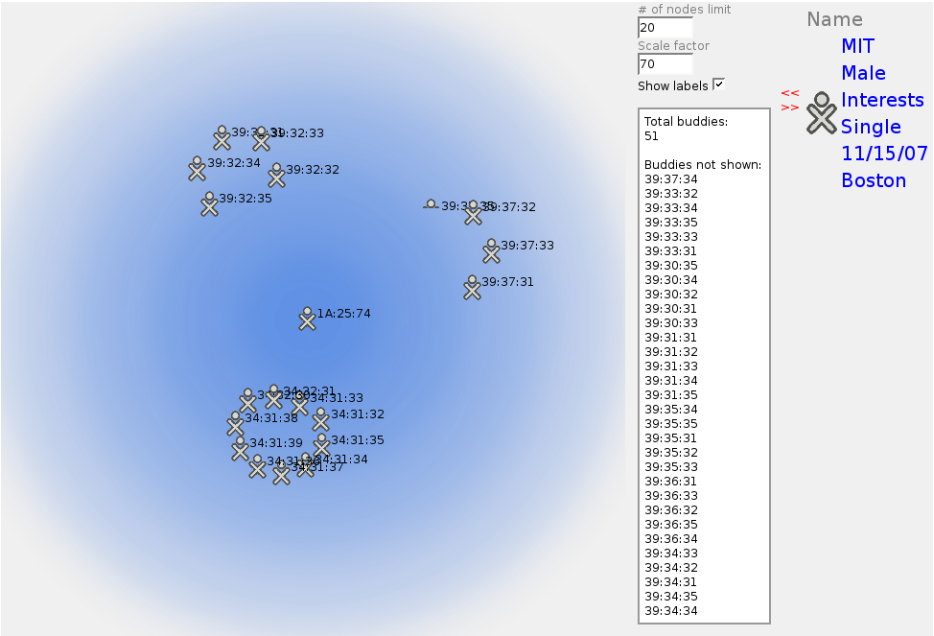


Figure 5-1: Web-based interface showing devices in physical proximity. The current device is placed at the center of the interface.

5.1.2 GTK-based UI

In order to make the next UI more efficient, we used GTK to provide a compiled, native interface. Although not as portable across different platforms as a web interface would be, GTK is faster and proved to be

[1] Implementation is attributed to David Gauthier *gauthier@media.mit.edu*

significantly easier to prototype new interfaces with. This user application communicates with Cerebro over DBus and has been tested x86-based Linux distributions, the Nokia N810 internet tablet [53] and the OpenMoko Freerunner mobile phone [58].

This UI involves four basic views; the user can switch between the four views using the buttons at the top. Each view offers a different set of buttons at the bottom of the window.

1. "Who" view: Shows a list of all devices in physical proximity, including their picture and nickname as reported in their node profile and distance as calculated in Section 3.2.9. Two options are provided at the bottom, a button to view a user's profile and a button to start a chat session with a user selected from the list (Figure 5-2a).

2. "What" view: A list of the profiles of all nodes in physical proximity. Two sub-views are provided: The first provides a sortable and searchable list of all profiles present in the network (Figure 5-2b) and the second allows the user to modify her profile by adding/removing entries from it (Figure 5-2d). Each entry in her profile is a key/value pair and certain key values are reserved, as shown in Table 5.1.

3. "How" view: An application that provides a means of communication with other nodes in the network. Two examples are shown in Figure 5-2: Figure 5-2c shows a simple chat session, while Figure 5-3 shows a game that involves bidding for items while chatting with other users. The "How" view provides different options depending on the application running on this view.

4. "When" view: A history of events such as nodes connecting and disconnecting from the network, user-defined alerts going off, etc.

5.1.3 OLPC: "Space" activity

The first attempt to build an application based on Cerebro was the "Space" activity for the Sugar environment [14]. Space retrieves a list of other XO laptops in the network and displays them on the plane based on their

Key	Description
`pic`	Path to a picture that will be used as the node's avatar.
`fileX`	Path to a file to be shared as part of the node's profile. "X" stands for the index number of the file, e.g. "file1", "file2", etc.

Table 5.1: Description of reserved key names in a user's profile.

respective distance from the current node, as shown in Figure 5-4. Space is similar to the web-based UI shown in Section 5.1.1, except that it does not exhibit the polling overhead of the web-based approach. The current node is placed at the center of the window and nodes are placed on a two-dimensional plane at a distance from the center proportional to the respective distance reported by Cerebro and at a random angle.

5.2 Next steps in discovery and context-aware applications

So far we presented only basic applications of user discovery and communication, without utilizing Cerebro's powerful event features. We can utilize event notifications such as a specific node or group of nodes joining or leaving the network, nodes coming close and becoming direct neighbors or moving away and events about one or more nodes changing their profile based on specific criteria (eg. an alert when an internet connection gets shared by a neighbor).

This section touches on future directions for applications that build on discovery, context-awareness, communication and group establishment for devices in the same physical area. For each of the following application areas, we present prior work and ways that Cerebro-based applications can provide an improved and richer user experience.

5.2.1 Friend Alerts

Being alerted when a friend is close-by in an urban environment is a well-known application. Existing approaches [] are based on mobile phones and/or require an internet connection to report the user's location to a server on the internet. Different methods are employed to infer position such as GPS, wifi ESSIDs and operator towers IDs. On the other hand, a Cerebro-based application does not require any of the above because discovery is performed directly among user devices, without requiring an internet connection or a mobile operator. An internet connection could be utilized to further improve user experience by accessing remote mesh networks over the internet (e.g. alert me when my friends go to our favorite pub in my home country, while I am abroad). Cerebro-based applications are also a better fit in environments where an internet connection is too expensive or not available, such as when visiting a foreign country: Cerebro makes it straightforward to discover users speaking the same language, having the same interests, etc.

5.2.2 Barter Economies (The Parking Problem)

In a barter economy goods or services are traded or exchanged without using money. By editing her profile, a user can provide a list of items and/or services she is willing to trade. Although it is hard to imagine users exchanging fruits and food anymore, it would make sense to bid for parking spots using an application similar to that shown in Figure 5-3. In dense urban environments where parking is usually a hard problem, the best parking spots are those that are about to become vacant right in front of us. A Cerebro-based application would provide not only an easy way to get access to just-in-time information, but also a means to form a small economy by bidding for the best spots. Increasingly more cars offer different connectivity options such as GSM and bluetooth, so it is reasonable to expect in the future cars to have standard WiFi and offering a similar service. A similar effort [68] to discover and distribute information about vacant parking spots will cost about $95M to establish the necessary infrastructure and will cause the cost of parking to increase even further. On

the other hand, the solution we propose will allow the users themselves to drive prices based on what other users are willing to pay to reserve a parking spot, as opposed to how much some sensor infrastructure has cost.

5.2.3 Hyper-local Advertisements

Similarly to the internet itself, Cerebro provides an open communication system by establishing a mesh network in which anyone can participate and can may attempt to contact or push data to anyone else. Building further on the previous example about discovering and bidding for parking spots, third parties can discover users bidding for parking spots to whom they can attempt to push advertisements of local parkings. This will provide parking companies with a highly targeted potential customer base.

The other facet of Cerebro being an open system is that each user has complete control over her device, choosing what data to keep and/or forward and for/from which other users. While any third party can attempt to push advertisements to one or more users in the mesh network, it is up to each receiver to choose whether to accept or reject unsolicited messages in some context, such as while bidding for a parking spot.

5.2.4 Ride-sharing

Increased traffic in urban environments has led to commuting alternatives such as ride-sharing. Again, similar efforts [12] to connect users willing to share a ride have been based on a centralized service, thus having poor just-in-time properties; users need to plan their trip well ahead in order to coordinate with other users.

We attempt to team up users as theaters and concerts let out hundreds of people, grouping users by destination. Indeed, a Cerebro-based application could allow users to state their destination in their profile and then it is up to their application to attempt to team up users together. Thus, we reduce the problem of ride-sharing in such a dense and time-critical scenario to

choosing the right key in the node's profile (eg. "ride-share") and efficiently grouping users together by destination.

5.2.5 Residential Bulletin Boards

A recent study [40] showed that while the internet may encourage communication across great distances, it may also facilitate interaction in a residential neighborhood. From our home computer we can discover and communicate with people from all over the planet, but there is no easy way to establish communication with people living in the same apartment building, other than going door-to-door. There are several efforts to create "e-neighborhoods" [19], but they work based on a website that acts as the rendez-vous point where users can search for potential neighbors. However, the existence of multiple such websites cause fragmentation which renders discovering the right website part of the neighbor discovery process itself.

A Cerebro-based application eliminates the need for "round-trip" communication through some website, enabling neighbors to directly discover each other and establish communication. Furthermore, the same Cerebro-based discovery process is used across different contexts —as shown in previous application examples—, thus adding more value to using Cerebro in each instance individually.

5.2.6 Networking in Trains, Airplanes and Traffic Jams

The common characteristic between traveling in a train, airplane, or being stuck in a traffic jam is that a number of strangers spend a significant amount of time together and are usually isolated from the rest of the world. Although their degree of isolation is constantly decreasing as internet connectivity becomes more ubiquitously available, they are still unable to discover and communicate directly with each other.

Airlines usually offer group games, such as poker and chess, that can be played among passengers, but a Cerebro-based application would run on the

user's computer and could afford a rich collection of games and disruptive applications to flourish, such as file-sharing among vehicles in a traffic jam or moving in the same direction on a highway.

5.2.7 The "Buy-1-Get-2" Deal

A challenging, especially in developing countries, is increasing our buying power. Bulk and wholesale orders can achieve better prices, but require additional capital. Lack of capital pushes developing societies into a poverty cycle where decreased consumption causes the cost of products to increase. We propose a discovery mechanism that will allow users to combine their buying power in order to place bulk orders and achieve better prices. In some cases of remote areas, mere shipment cost is prohibiting access to basic products.

A mobile phone-based solution offering direct discovery and communication could help users form purchase-driven communities in an effort to break the poverty cycle. Users in supermarkets and wholesale stores could also benefit from such an application by engaging in joint transactions. For example, by sharing a "buy-1-get-2" deal, they purchase a product at the quantity they really need at a wholesale price. All they need to do is modify their profile to make their intent explicit to other users in order to initiate the discovery process.

This application would be beneficial to the wholesale store or supermarket itself. Similarly to the advertisement example in Section 5.2.3, the store could seed the discovery process among users by making wholesale deals available in the mesh network (as part of the store's profile) and allow the users to easily browse and click on deals and fill empty spots in a combined purchase.

5.2.8 Dynamic Bus Routes

Being a means of mass transportation, buses need to have fixed schedules and routes that attempt to service as many passengers as possible. A

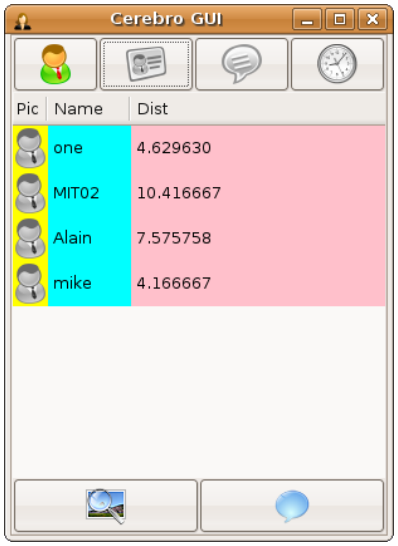
Cerebro-based discovery an communication system between passengers, bus-stops and buses could enable a bus to make informed, weighted decisions on both its schedule and route. While we can maintain existing stops and timetable, we can dynamically adapt the route between stops and dynamically increase runs to match passenger demand. Users can modify their profile to state their destination address and the bus will weight requests among passengers against rerouting costs in terms of time and energy. Passengers may even choose to tip the bus's decision in their own interest by increasing their fare.

The proposed application for dynamic bus routes would provide a more fine-grained service to passengers, thus paving the way for a transportation service in between a taxi and a bus.

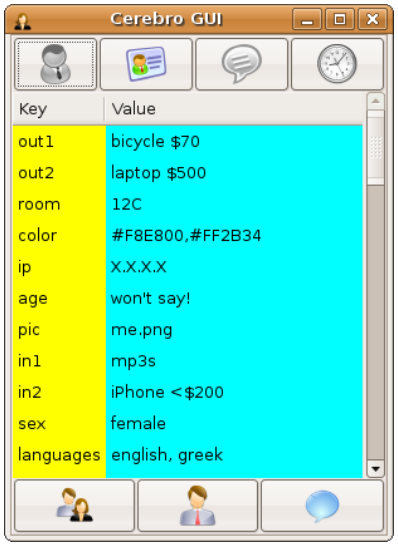
5.2.9 Emergency and Security

Cerebro builds an ad-hoc network and does not require any infrastructure, providing better connectivity among mesh network participants than to the internet at large. This makes Cerebro ideal for emergency applications in which we need to raise awareness of an event among network participants rather than solely relying on a centralized service. For example, discovering the presence of a medical doctor in physical proximity may prove critical in a medical emergency, instead of relying solely on the fast response of an ambulance. A Cerebro-based emergency application could further build on Cerebro's history by contacting a doctor that is no longer in physical proximity, but was present a few minutes ago.

MIT recently adopted an emergency notification system [18] that maintains contact information of all MIT community members to be notified in an emergency situation. Instead of running the risk of ignoring MIT visitors, a Cerebro-based application could notify all users in physical proximity, irrespective of their affiliation with MIT.

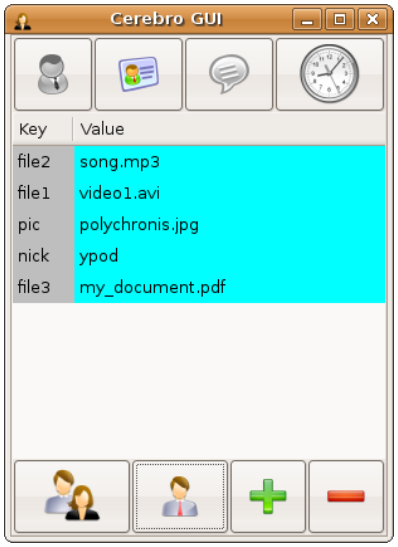


(a) "Who" view: List of devices in proximity, showing the picture, name and distance for each device.



(b) "What" view: Node profile of an example device.

(c) "How" view: Chat session with neighbors.



(d) Modifying the local profile: The user has set her nickname, her picture and is sharing three files.

Figure 5-2: GTK-based UI. The first three Figures are the first three panels of the UI used on OpenMoko Freerunner. In Figure 5-2d, the user modifies her profile by clicking on the single-person button at the bottom of the window.

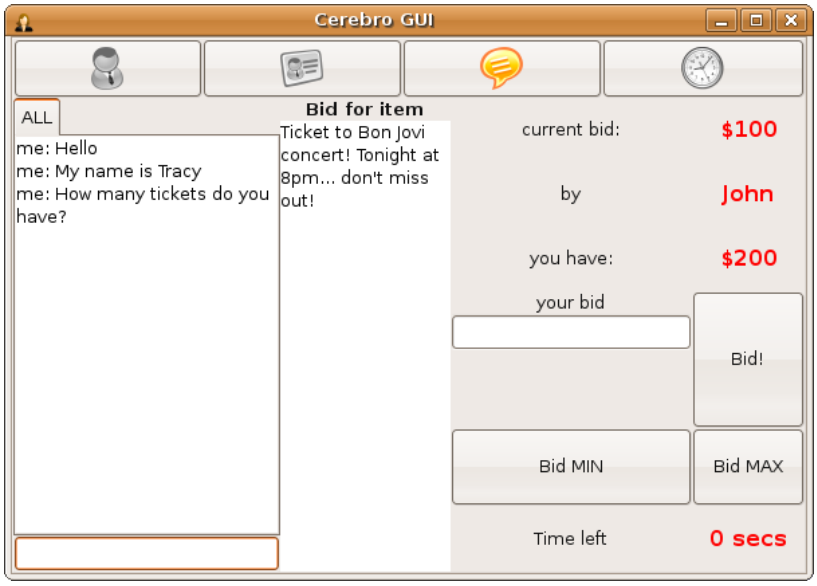


Figure 5-3: The figure is based on a prototype for the Nokia N810. The bidding game provides a chat session while users bid for an item.

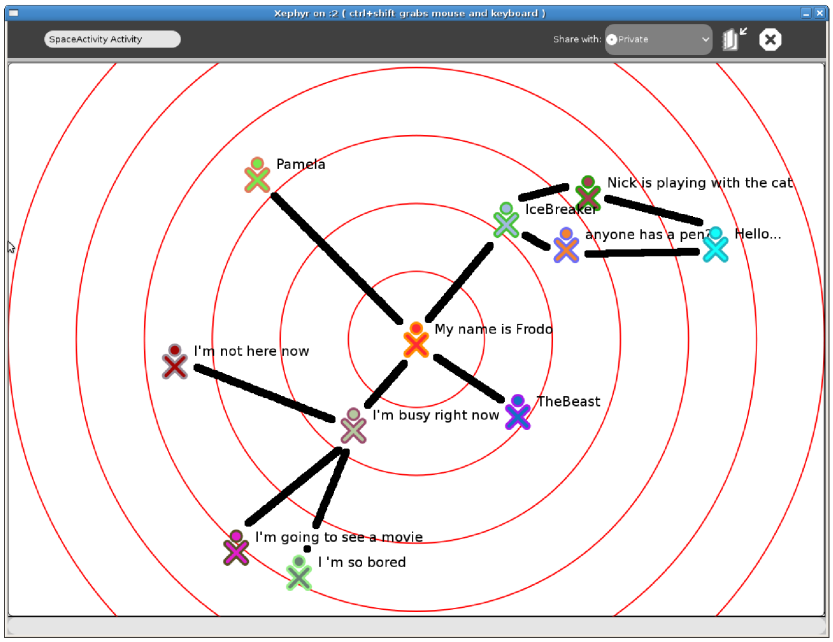


Figure 5-4: Snapshot of "Space" activity for the Sugar environment.

Chapter 6

Evaluation

In this chapter, we evaluate the performance of Teleporter and Cerebro's presence protocol.

6.1 Experimental results

We evaluate Cerebro's implementation in terms of its performance and scalability properties. First we evaluate Teleporter's performance when transferring a 2MB file to 27 neighbors. We use TCP/IP as the baseline of our measurements.

Then, we analyze the network overhead as the aggregate amount of traffic imposed by the presence protocol against the time it takes for all 65 nodes in a fully connected mesh network to discover each other and share their profile.

6.1.1 Teleporter's performance

In order to evaluate Teleporter, the module responsible for performing parallel data transfer among neighbors, we set up a network of 28 nodes in which one node needs to transfer a 2MB file to the rest of the nodes. We used OLPC XO laptops (hardware version: B4) that were layed out as shown

in figure 6-1. We did not make provision for providing a clear environment from an RF perspective. On the contrary, the experiment was carried out at an environment that is considered noisy by its occupants.

There were 12 nodes placed next to each other in the “Library area” (node names starting with “L”). Another 7 nodes were placed next to each other in “Richard’s area” (node names starting with “R”) and 8 nodes were hanging from the ceiling in the “Garden area” (node names starting with “C”). The sender as placed close to the first 12 nodes in the “Library area”.

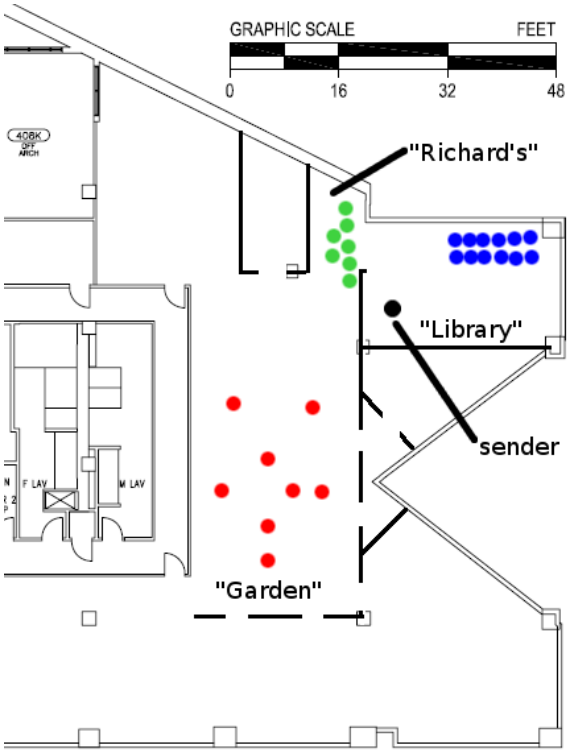


Figure 6-1: Testbed layout in space. Blue-colored nodes are in the “Library area”, green nodes are in “Richard’s area” and red nodes are in the “Garden area”.

For each node we measured the amount of data received as a function of time. Figure 6-2 shows data received at each node (in bytes) as a function of time. Each of the three groups of nodes is shown in a different color. The transfer was started by sending the file to a node in the “Library area” first. This is also evident in figure 6-2, as the first reception that completes is a blue line.

By the time that the first reception completes, at least 80% of the data is

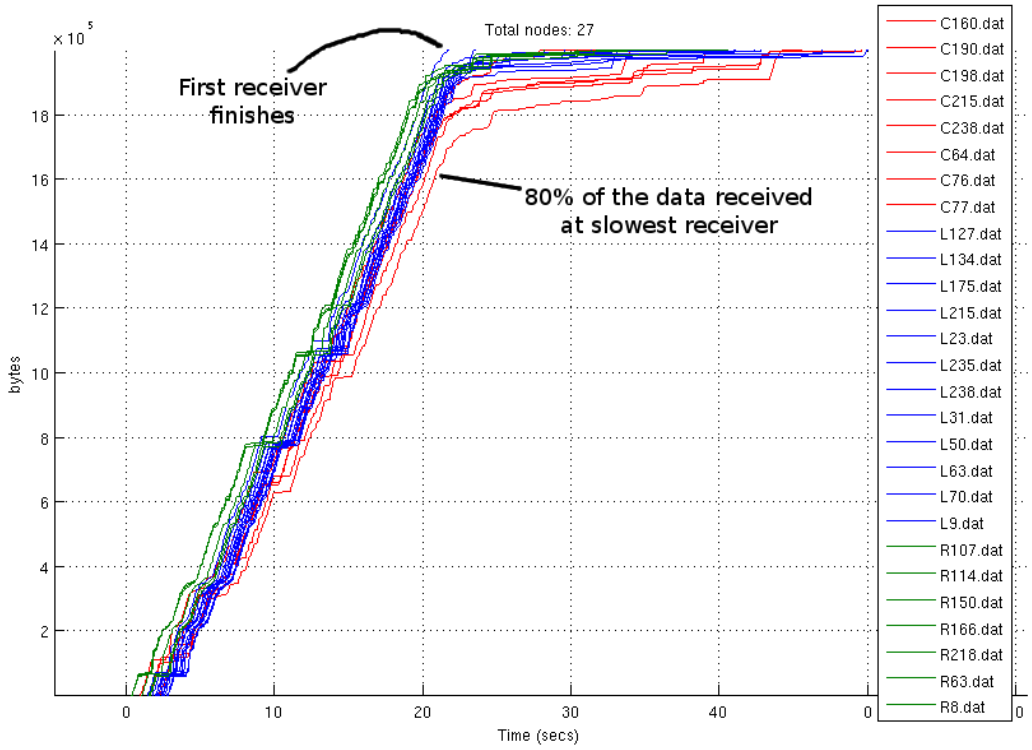


Figure 6-2: 2MB transfer to 27 nodes.

received on all receivers. This implies a massive reduction in the total amount of traffic required to transfer the data to all receivers. Every time it finishes transmitting the data to one receiver, Teleporter will randomly choose the next receiver from a given set of destinations, and will only transfer the data pieces that the new receiver is missing.

The first transmission completed in about 20 seconds, with an average throughput of 0.8Mbit which is close to the nominal broadcast rate of 1Mbit. However, it took the last receiver and additional 30 seconds to receive the remaining 20% of the data. This is because Teleporter attempts to send data as fast as possible by blasting bursts of data and then waiting for a NACK response from the receiver. The sender will keep sending frames at small time intervals (d msecs, as described in Section 3.3.2) until it receives a NACK or a timeout elapses. This requires some sort of synchronization with the sender in order to make sure it receives NACKs when it actually expects them; sending a NACK to the sender amidst a burst of data makes it unlikely that it will receive it. Such lack of synchronization results the "long tail" effect

shown in figure 6-2, while sending missing pieces of data to consecutive receivers. This effect reduces the sender's effectiveness by forcing it to spend more time waiting for a NACK (during the time intervals d), rather than sending bursts of data. We expect this issue to be addressed in the next version of Teleporter, by adopting a well-known technique like TCP's "slow start" to ensure better synchronization between the sender and the receivers.

In order to obtain a better view of Teleporter's performance, we compare it against an ideal TCP data transfer for the same data and channel conditions. After the Teleporter experiment, we attempted multiple TCP transfers to various destinations in the network shown in figure 6-1 and chose the best throughput obtained over all destinations to be used as the baseline for Teleporter's evaluation. Although the best throughput obtained does not generalize linearly for the aggregate throughput we would get from a TCP transmission to each destination, we use it to demonstrate that Teleporter's performance is consistently superior to TCP and that we are not observing an artifact.

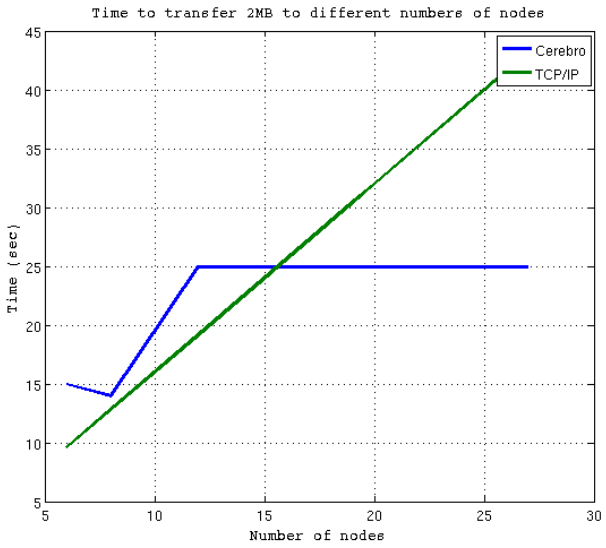


Figure 6-3: Total time for TCP vs. Teleporter to transfer a 2MB file as a function of the number of destinations.

Figure 6-3 shows the total time necessary to transfer 90% of a 2MB file as a function of the number of destinations. Teleporter used a network capacity of

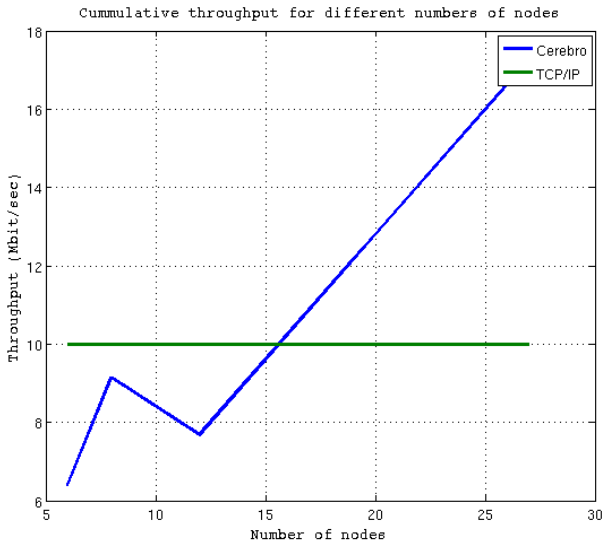


Figure 6-4: Aggregate throughput for Teleporter vs. TCP while transferring a 2MB file as a function of the number of destinations.

1MBit (broadcast rate) per destination, while the best performance obtained with TCP was 10MBit, a speed that is 10-times better than the nominal that was available to Teleporter. As a result, Teleporter by definition will not be able to surpass TCP's aggregate performance for at least the first 10 destinations. Indeed, Teleporter's outperforms TCP only after the first 15 receivers. In other words, if the 2MB file is to be transferred to less than approximately 15 nodes, TCP will perform better than Teleporter. However, for more than 15 receivers, Teleporter's time to transfer 90% of the file remains constant, while for TCP it keeps increasing.

Similarly, Figure 6-4 shows the aggregate throughput of Teleporter vs. TCP as a function of the number of receivers: Teleporter's throughput increases linearly, while for TCP it remains the same.

6.1.2 Presence algorithm's performance

Traffic measurement

We evaluate the presence protocol in terms of its traffic overhead and responsiveness to "join" events in the network. In order to stress the

algorithm's performance as much as possible, we evaluated it on a 65-node test-bed, again using OLPC XO laptops (hardware version: Mass Production).

All 65 nodes were dispersed in the "Garden area" that is shown in figure 6-1, forming a fully connected network. Cerebro was started on all nodes in less than 5 seconds and the generated traffic was monitored using Wireshark[1]. Figure 6-5 shows a snapshot of the captured packet trace [10].

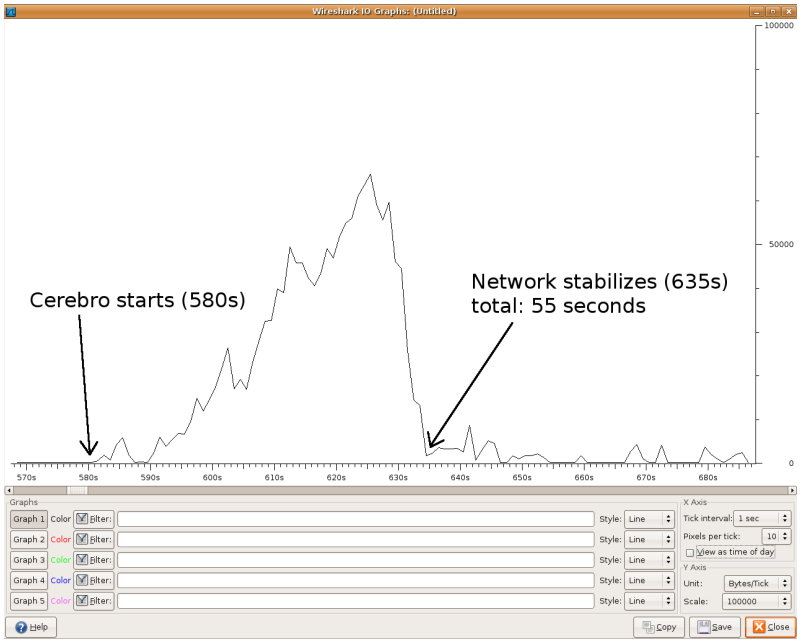


Figure 6-5: Snapshot of a packet trace while 65 nodes were running Cerebro.

A fully connected network where all nodes start Cerebro at the same time maximizes the network traffic spike due to nodes exchanging presence information and node profiles. Each node shared a 133-byte profile and therefore the total amount of bytes to be exchanged in the network was:

$$65 * 64 * 133 = \boxed{553{,}280 \text{ bytes}}$$

because each of the 65 nodes must send 133 bytes to the other 64 nodes. We measured the total amount of traffic regarding node profiles exchange (ethernet protocol type `0x7600`) that was sent between the 580th and 635th second, as shown in figure 6-5 and the measurements are summarized in

[1] http://www.wireshark.org/

Table 6.1. The total capture size (1,158,111 bytes) is double the total traffic (553,280 bytes) that should have been produced. In other words, on average, each node profile was sent twice per destination and all node profiles were reliably transferred to all nodes in the network.

Total capture size	1,158,111 bytes
Average load	20 kbytes/sec
Average packet size	147 bytes
Total capture time	53 seconds

Table 6.1: Traffic measurement regarding node profile exchange (ethernet protocol type $0x7600$) from the captured packet trace [10] during seconds 580 and 635.

We should note that if TCP was used instead of Teleporter for transferring node profiles, for every TCP retransmission, 9 retransmissions would take place at layer-2, as per 802.11 protocol for unicast transmissions. In a dense environment with 65 nodes in ad-hoc mode, one would expect multiple TCP retransmissions (each responsible for 9 layer-2 retransmissions) per node profile. Instead, Teleporter achieved the same result with a total of only 2 layer-2 retransmissions. Teleporter achieved this result because of its parallel data transfer ability.

Time measurement

In order to get a better insight on the time necessary for all 65 nodes to become aware of the presence and node profile of each other, we measured the time offset for each profile reception after having received the first profile.

Figure 6-6 shows the number of profile receptions —in the form of "arrivals"— as a function of time. Each bar is a 3-second bucket showing the average number of "arrivals" for the specific bucket among all 65 nodes and the blue lines show the standard deviation.

We observe that it took less than 55 seconds in total for all 65 nodes to receive presence information from everyone else, in accordance to figure 6-5 showing that the spike in network traffic lasts about 55 seconds. However, it is evident that most of the "arrivals" took less than 30 seconds.

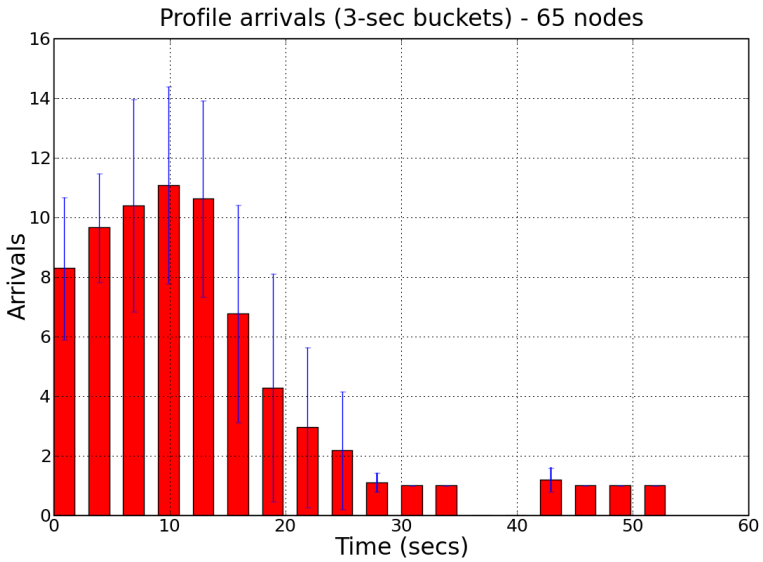


Figure 6-6: Distribution of node profile arrivals: The bars are 3-second buckets showing the average number of arrivals per bucket over all 65 nodes; the lines show the standard deviation.

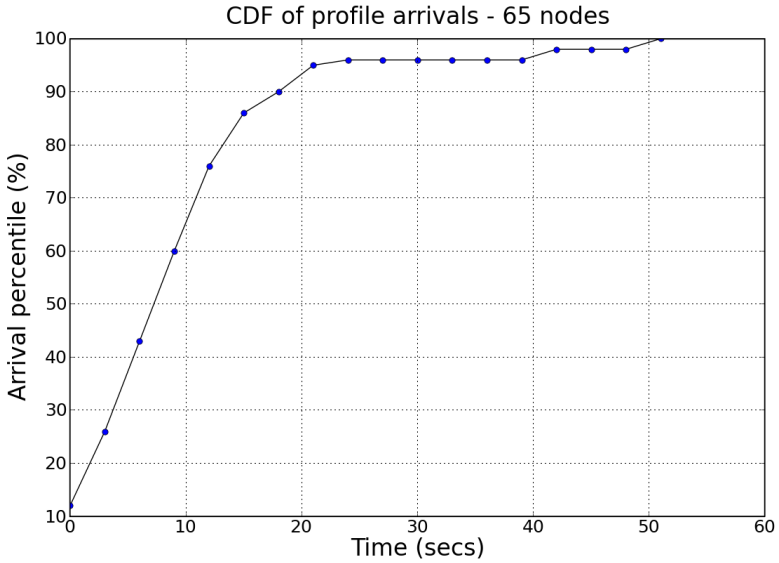


Figure 6-7: Cumulative distribution function (CDF) of node profile arrivals: 90% of all profiles arrived in less than 20 seconds.

Indeed, figure 6-7 shows the cumulative distribution function (CDF) of profile arrivals. We observe that 90% of all node profiles arrived in less than 20 seconds.

6.2 Hardware

Cerebro has been tested on various hardware platforms, as shown in Table 6.2

Platform	CPU	OS
PC	x86	Ubuntu/Fedora
OLPC XO	x86	Fedora
Nokia N810	ARM	Linux-Maemo
OpenMoko FreeRunner	ARM	embedded Linux
Gumstix	ARM	embedded Linux

Table 6.2: Hardware on which Cerebro has been tested.

6.3 Discussion

Experience from working with OLPC has shown that designing and implementing a service that provides presence information in real-world scenarios is not just about performance and scalability. This power-saving scheme of the device on which Cerebro runs, may be as aggressive as placing the CPU in sleep mode regularly for more than a second at a time. By adapting to the density of the network, Cerebro's presence protocol ensures that, on average, the device will receive at most one presence beacon from its neighbors every $\frac{1}{\rho}$ seconds, thus placing the algorithm's wake-up events on a regular schedule that conforms with the device's power-saving scheme.

In terms of scalability, Cerebro can accommodate up to 100 nodes per presence beacon—each presence beacon is up to 1500 bytes long and each presence entry is 15 bytes. By adjusting $\rho = \frac{1}{3}$ —one beacon every 3 seconds—, the presence protocol's traffic overhead is 500 bytes/sec for every 100 nodes.

We chose to adapt the period τ for sending presence beacons to be proportional to the total number of nodes N ($\tau = \frac{N}{\rho}$), in order to strike a balance between the traffic overhead and the algorithm's responsiveness to network events, so they are both proportional to $O(N)$. If we chose to adjust τ according to $\tau = \frac{N^2}{\rho}$, the traffic overhead would be constant regardless of

N, but the algorithm's responsiveness would be poor and proportional to N^2.

Chapter 7

Conclusion

This thesis has presented the design, implementation and evaluation of Cerebro, a system that allows humans and objects in physical proximity to discover each other and share data and services. Cerebro brings closer a future where humans will be able to discover and interact with their physical environment as part of a mobile network that Cerebro creates and could span thousands of participants.

From a technical perspective, mesh/ad-hoc networks and data transport protocols have traditionally been considered as two separate domains: The first deals primarily with the aspects of routing in order to keep the network connected, while the second is concerned with transferring data from some source to some destination. Cerebro provides presence information by blending aspects from both domains in order to maximize their collective utility.

Scalability through simplicity is Cerebro's core design principle. Based on this principle, Cerebro implements three innovative features that have significant impact on the discovery and communication in a mesh network:

- **Node profiles**. Each node provides information about the data and services it is sharing in the form of key/value pairs. Node profiles act as a list of pointers to the data and services that each node is sharing. This structure allows node profiles to be short but comprehensive and extensible (see also section 3.3.1).

- **Dynamic presence protocol**. The presence protocol exhibits linear network traffic overhead with the total number of nodes in the network by adjusting the rate at which presence beacons are sent out in order to accommodate a varying local network density. As a result, the traffic overhead is independent of the layout of the network (see also section 3.2.1).

- **Parallel data transfer**. Teleporter was designed to perform reliable data transfer in one-to-many scenarios where the receivers are direct neighbors to the sender (see also section 3.3.2). As the number of receivers grows, the time-to-completion for TCP-based solutions increases linearly, while for Teleporter it is almost[1] constant (see also section 6.1.1). Contrary to other approaches, Teleporter's efficiency increases with the number of destinations.

Finally, from an application perspective, the combination of routing and data transport is traditionally used for internet access, whereas Cerebro additionally powers applications whose goal is to stay in the mesh network than to get out of it. The value of a network increases with the number of participants [27] and Cerebro's scalability properties allow applications to draw value by communicating with other mesh network participants, rather than focusing solely on sharing an internet connection.

7.1 Implications and Opportunities

7.1.1 Extreme presence information: Scaling in the 1000s

A crucial part of the vision behind Cerebro is to provide presence information on a city-scale that may involve thousands of nodes. Clearly, it is unrealistic to attempt to deterministically provide presence information about all the nodes in such a network.

One way to approach this challenge would be to limit automatic propagation of node profiles in the network up to a specific radius in order to ensure an

[1] Time-to-completion is constant for about 90% of the data to be sent, while section 7.1 proposes some techniques that deal with the remaining 10%.

upper limit on Cerebro's proactive traffic. Bloom filters [21] could be used to summarize the presence of nodes beyond the pre-defined radius. Bloom filters have already been proposed in routing [47][35]. One way to determine the radius is to limit the number of presence entries that each node is advertising in its presence beacon(s) to only the closest[2] M nodes, while summarizing information about nodes outside the effective radius in a Bloom filter, as shown in figure 7-1. A shortcoming of this approach is that the filter's accuracy in responding to queries will decrease as it propagates throughout the network and accumulates presence information.

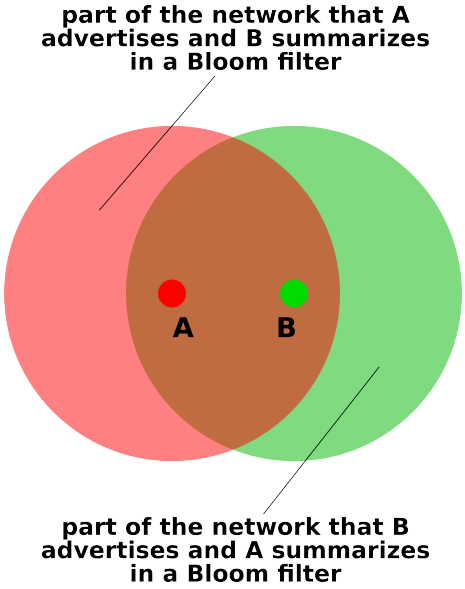


Figure 7-1: Summarizing information outside the node's effective presence information radius in a Bloom filter.

Additionally, a publish/subscribe [30] approach could be adopted to retrieve presence information from specific remote nodes in which the user maintains interest, as shown in figure 7-2. There could be different approaches regarding the incentives to motivate nodes outside the effective radius to route data for two or more remote nodes. One option would be to engage in a transaction with every node, such as "I will share with you my internet connection if you route for me". Another option would be to base routing outside the effective radius on the social network of either end of the data being exchanged. For example, "I will propagate presence information

[2] Distance information is already provided by Cerebro's presence protocol.

pertaining to nodes inside my effective radius, or for friends and family no matter where they are".

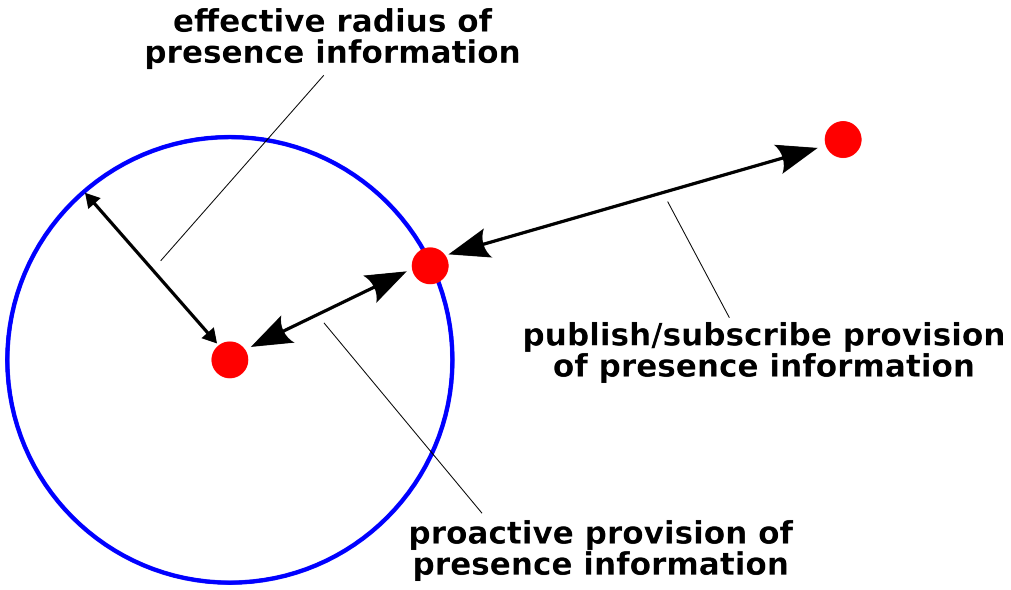


Figure 7-2: Limiting proactive distribution within a specific radius, while using a publish/subscribe mechanism to retrieve presence information from outside this radius.

7.1.2 Dedicated hardware for presence information

Cerebro is meant to be used primarily with a device that humans can carry all the time. Running Cerebro on general purpose hardware, such as a laptop computer is inefficient because it requires the device to be always on. Using mobile phones is also suboptimal because, at best, they offer an 802.11-based radio and/or Bluetooth to be used for local discovery and data exchange and their use is not power-efficient.

Using dedicated hardware to implement Cerebro's functionality would allow us to exercise aggressive power saving by using a low bandwidth but power-efficient radio for presence beacons, such as a ZigBee [42], while using a high-bandwidth, 802.11-based radio for sharing data and node profiles. As a result, we would benefit from the advantages of both types of radios, while minimizing power consumption.

Another advantage of the use of dedicated hardware would be to make it a USB device so that general purpose hardware could benefit from the services and applications that Cerebro offers, while keeping the device itself portable. There are already USB dongle devices [76] that combine a WiFi radio with storage and some minimal processing power, as shown in figure 7-3.

Figure 7-3: USB dongle that combines a WiFi radio with flash memory by TrendNet.

Appendix A

Cerebro Implementation Notes

A.1 Cerebro API

This section presents the DBus signals and methods that allow an application to access Cerebro's services. The latest version of Cerebro's API can be found at `http://cerebro.mit.edu/index.php/Documentation`

A.1.1 Signals

The following table summarizes the DBus signals that get emitted on different events.

Signal name	**Description**
`on_node_info_arrival`	Triggers when a new profile arrives, or an existing profile gets modified. Returns the profile and a flag that is true if the profile is from a new node and false if it is a modification to an existing profile.
`on_node_arrival`	Triggers when a new node appears in the network. This event is independent of having a profile about the new node.
`on_node_departure`	Triggers when a node is considered disconnected from the network.
`on_request`	Triggers when a request is received. Returns the content of the request and the ID of the application it is intended for. An application must be registered with Cerebro under the specific ID, otherwise the signal will not trigger.
`on_received_data`	Triggers when new data is received. Provides the ID of the application that the data is for and a flag to signal whether the intended destination is the current node.

A.1.2 Methods

The following table summarizes the DBus methods that can be invoked from an application.

Method name	Description
`register`	Register an the instance of an application with Cerebro. The instance gets shared with in the neighborhood. If there is another instance shared by one or more nodes, the new instance joins them in a "virtual room" and all instances can communicate with each other. If there is no other instance of the same application shared already, a new "virtual room" is created and joined.
`unregister`	Leave the "virtual room"
`get_node_id`	Returns the unique ID of the current node.
`get_pers_info`	Returns a dictionary that contains the user's profile.
`set_pers_info`	Sets the dictionary that contains the user's profile.
`list_acts`	Returns a list of all shared application instances.
`get_aid`	Returns the unique ID of the current application instance.
`show_mst`	Returns the minimum spanning tree rooted at the current node.
`show_dneighs`	Returns a list of direct neighbors.
`show_net_tree`	Returns a list of all known paths to all known nodes.
`push_data`	Pushes data to all nodes in the same "virtual room" as the current node. A buffer containing the data to be sent is passed as argument to this method.
`send_file`	Sends a file to all nodes in the same "virtual room" as the current node. `send_file` is essentially an abstraction to `push_data` that is used to send files. The path to the file is passed as argument to this method.
`push_to`	Sends data to a node under a specific unique ID. The node does not need to be in the same "virtual room" as the current node.

A.2 Calculating an IP address from a MAC address

The following sample Python code calculates an IP address in the `10.0.0.0/8` subnet by setting the last three bytes of the IP address based on the last three bytes of a MAC address that is provided as input. The last three bytes of the MAC address are unique with high probability, thus the resulting IP address is also unique with high probability. This probability is:

$$p > 1 - \frac{1}{(2^8)^3} > 0.9999 \tag{A.1}$$

```
def calc_IP(mac):
    if len(mac) != 17:
        return ''
    mac = re.split(':', mac)

    c1 = '10'
    c2 = str(int(mac[3], 16))
    c3 = str(int(mac[4], 16))
    c4 = str(int(mac[5], 16))

    return "%s.%s.%s.%s" % (c1, c2, c3, c4)
```

Bibliography

[1] Avahi. http://www.avahi.org/.

[2] Bonjour. http://bonjour.macosforge.org/.

[3] Context-based routing. http://serl.cs.colorado.edu/~carzanig/cbn/index.html/.

[4] Gumstix devices. http://www.gumstix.com/.

[5] Internet penetration in the us between 2000-2008. http://www.internetworldstats.com/am/us.htm.

[6] Jaiku. http://www.jaiku.com/.

[7] Jambonet. http://www.jambo.net/.

[8] Meetro. http://www.meetro.com/.

[9] multicast dns. http://www.multicastdns.org/.

[10] Packet capture during the 65-node presence algorthim experiment. http://cerebro.mit.edu/capture-1.

[11] Rakket. http://www.rakket.com/.

[12] Ride-sharing. http://www.erideshare.com/.

[13] Roofnet. http://pdos.csail.mit.edu/roofnet/.

[14] Sugar labs. http://www.sugarlabs.org/.

[15] Twitter. http://www.twitter.com/.

[16] Zero configuration networking. http://www.zeroconf.org/.

[17] *Cross-layer design for wireless networks*, volume 41, Oct 2003.

[18] MIT Alert. Emergency alert system. http://web.mit.edu/mit-emergency/mitalert/.

[19] UPenn Annenberg School of Communications. i-neighbors. http://www.i-neighbors.org/.

[20] Balakrishnan. Designing high-capacity wireless networks. http://nms.lcs.mit.edu/6.829-f06/lectures/L12-capacity.pdf.

[21] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[22] David Braginsky and Deborah Estrin. Rumor routing algorthim for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31, New York, NY, USA, 2002. ACM.

[23] Chroboczek. Babel routing protocol. http://www.pps.jussieu.fr/~jch/software/repos/babel/babel.text, 2008.

[24] Jacquet Clausen. IETF RFC3626: Optimized link state routing protocol (olsr). http://tools.ietf.org/html/rfc3626, October 2003.

[25] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 134–146, New York, NY, USA, 2003. ACM.

[26] D. Crockford. IETF RFC4627: The application/json media type for javascript object notation (json). http://www.ietf.org/rfc/rfc4627.txt?number=4627, July 2006.

[27] Reed DP. The law of the pack. *Harvard Business Review*, pages 23–24, February 2001.

[28] R. Droms. IETF RFC2131: Dynamic host configuration protocol (dhcp). http://tools.ietf.org/html/rfc2131, October 1996.

[29] Inside Higher ED. The brave new world of myspace and facebook. http://www.insidehighered.com/views/2007/04/03/steinbach.

[30] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[31] Fonseca, Gnawali, Jamieson, and Levis. Four-bit wireless link estimation. Technical report, UC Berkeley, Univ. of Southern California, MIT CSAIL, Stanford Univ., 2007.

[32] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, 2006.

[33] freedesktop.org. D-bus. http://www.freedesktop.org/wiki/Software/dbus.

[34] Lau R. Fuhr P. The realities of dealing with wireless mesh networks. http://wireless.sensorsmag.com/sensorswireless/Mesh/The-Realities-of-Dealing-with-Wireless-Mesh-Networ/ArticleStandard/Article/detail/318652.

[35] Robert Gilbert, Kerby Johnson, Shaomei Wu, Ben Y. Zhao, and Haitao Zheng. Location independent compact routing for wireless networks. In *MobiShare '06: Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, pages 57–59, New York, NY, USA, 2006. ACM.

[36] Omprakash Gnawali, Ki-Young Jang, Jeongyeup Paek, Marcos Vieira, Ramesh Govindan, Ben Greenstein, August Joki, Deborah Estrin, and Eddie Kohler. The tenet architecture for tiered sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 153–166, New York, NY, USA, 2006. ACM.

[37] Guardian. Facebook information should be regulated, survey says. http://www.guardian.co.uk/technology/2008/jun/05/privacy.socialnetworking.

[38] Gumstix. Gumstix embedded device. http://www.gumstix.com.

[39] Zygmunt J. Haas, Joseph Y. Halpern, and Li Li. Gossip-based ad hoc routing. *IEEE/ACM Trans. Netw.*, 14(3):479–491, 2006.

[40] Keith N. Hampton. Neighborhoods in the network society: The e-neighbors study. *Information, Communication, & Society*, 10(5):714–748, 2007.

[41] Golder Wilkinson Huberman. Rhythms of social interaction: messaging within a massive online network. http://www.hpl.hp.com/research/idl/papers/facebook/facebook.pdf.

[42] IEEE. Zigbee: IEEE 802.15 WPAN Task Group 4 (TG4). http://www.ieee802.org/15/pub/TG4.html.

[43] Y.G. Iyer, S. Gandham, and S. Venkatesan. Stcp: a generic transport layer protocol for wireless sensor networks. *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on*, pages 449–454, Oct. 2005.

[44] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. *SIGCOMM Comput. Commun. Rev.*, 36(4):243–254, 2006.

[45] Sukun Kim, Rodrigo Fonseca, Prabal Dutta, Arsalan Tavakoli, David Culler, Philip Levis, Scott Shenker, and Ion Stoica. Flush: a reliable bulk transport protocol for multihop wireless networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 351–365, New York, NY, USA, 2007. ACM.

[46] Philip Levis, Neil Patel, Scott Shenker, and David Culler. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI*, pages 15–28, 2004.

[47] Xiuqi Li, Jie Wu, and Jun Xu. Hint-based routing in wsns using scope decay bloom filters. *Networking, Architecture, and Storages, 2006. IWNAS '06. International Workshop on*, pages 8 pp.–, Aug. 2006.

[48] Cameron Marlow. *The Structural Determinants of Media Contagion.* PhD thesis, MIT, 2005.

[49] C. Mbarushimana and A. Shahrabi. Comparative study of reactive and proactive routing protocols performance in mobile ad hoc networks. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 679–684, Washington, DC, USA, 2007. IEEE Computer Society.

[50] P. Mockapetris. IETF RFC1035: Domain names: Implementation and specification. http://tools.ietf.org/html/rfc1035, November 1987.

[51] BBC News. Craigslist's silent emergence. http://news.bbc.co.uk/1/hi/business/4724165.stm.

[52] Nielsen. Social networking going mobile. http://www.nielsen.com/media/2008/pr_080508.html.

[53] Nokia. Nokia n810 internet tablet. http://nokia.us/A4626059.

[54] Christian Nold. Biomapping. http://www.biomapping.net/.

[55] OLPC. http://www.laptop.org/.

[56] OLPC. Bug report on usage of mdns in a mesh network. http://dev.laptop.org/ticket/6162.

[57] OLPC. Collaboration network testbed. http://wiki.laptop.org/go/Collaboration_Network_Testbed.

[58] OpenMoko. Openmoko. http://www.openmoko.org.

[59] Jorge Ortiz, Chris R. Baker, Daekyeong Moon, Rodrigo Fonseca, and Ion Stoica. Beacon location service: a location service for point-to-point routing in wireless sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 166–175, New York, NY, USA, 2007. ACM.

[60] OSI. OSI. `http://www.osi.org`.

[61] A. Lippman M. Bletsas P. Ypodimatopoulos, D. Reed. Presence information in mobile mesh networks. IEEE Consumer Communications and Networking Conference 2008.

[62] Jeongyeup Paek and Ramesh Govindan. Rcrt: rate-controlled reliable transport for wireless sensor networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 305–319, New York, NY, USA, 2007. ACM.

[63] Govindan R. Paek J. Rcrt: Rate-controlle reliable transport for wireless sensor netoworks. In *SenSys*, 2007.

[64] Eric Paulos. `http://www.urban-atmospheres.net`, 2004.

[65] C. Perkins. IETF RFC2002: Mobile ip. `http://tools.ietf.org/html/rfc2002`, March 1997.

[66] Bhagwat P. Perkins C. Highly dynamic destination-sequenced distance vector routing (dsdv) for mobile computers. In *SIGCOMM*, 1994.

[67] Aaron Quigley. Large scale force directed layout. `http://www.cs.usyd.edu.au/~aquigley/3dfade/`.

[68] Techonology Review. Find a parking space online. `http://www.technologyreview.com/Infotech/21123/?a=f`.

[69] Bor rong Chen, Kiran-Kumar Muniswamy-Reddy, and Matt Welsh. Ad-hoc multicast routing on resource-limited sensor nodes. In *REALMAN '06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 87–94, New York, NY, USA, 2006. ACM.

[70] MIT Senseable City Lab. Wikicity. `http://senseable.mit.edu/wikicity/rome/`.

[71] Mao Wang Qiu Lam Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *Usenix*, 2007.

[72] SniffLabs. Snifftag. `http://www.sniflabs.com/`.

[73] F. Stann and J. Heidemann. Rmst: reliable data transport in sensor networks. *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 102–112, May 2003.

[74] A. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.

[75] OLPC Trac. Ticket 5078: A more mesh-friendly presence protocol for salut. `http://dev.laptop.org/ticket/5078`.

[76] TrendNet. Usb adapter w/ wifi 802.11b/g w/ 512mb storage. `http://trendnet.com/products/proddetail.asp?prod=150_TEW-429UF&cat=84`.

[77] Schiller Voisard. *Location-based Services*. Morgan Kaufmann, 2004.

[78] Whrrl. Location and event bookmarking and sharing with social network using mobile phones. `http://www.whrrl.com/`, 2008.

[79] Levis Brewer Culler Gay Madden Patel Polastre Shenker Szewczyk Woo. The emergence of a networking primitive in wireless sensor networks. *Communications of the ACM*, 51:99–106, 2008.

[80] Culler D. Woo A., Tong T. Taming the underlying challenges of reliable multihop routing in sensor netoworks. In *SenSys*, 2003.

[81] G. Riley X. Zhang. Scalability of an ad hoc on-demand routing protocol in very large-scale mobile wireless networks. `http://sim.sagepub.com/cgi/reprint/82/2/131.pdf`.

[82] XMPP. Extensible messaging and presence protocol. `http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol`, 2008.

[83] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor network for structural monitoring. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24, New York, NY, USA, 2004. ACM.

[84] P. Ypodimatopoulos. A 65-node experiment using cerebro. `http://cerebro.mit.edu/index.php/65-node_test`.

[85] Floyd Jacobson Liu McCanne Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking (TON)*, 5:784 – 803, 1997.