Middleware for Context-Aware Opportunistic Networks

PANTELEIMON PANIDIS

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2006

COS/CCS 2006-1

Middleware for Context-Aware Opportunistic Networks

Panteleimon Panidis

School of Information and Communication Technology
Royal Institute of Technology (KTH)
Stockholm, Sweden

18/01/06

Abstract

Mobile devices such as palmtops and cell phones are continuously increasing in capabilities and popularity. At the same time, due to their decreasing price they are becoming more and more attractive and available to the average customer. This has lead to the development of many new applications for such portable electronic devices. Mobile devices tend to have increasing local resources in terms of memory/storage and CPU [2]. Despite these improvements in hardware attributes, there are still limitations that characterize these portable devices and which have not improved as quickly as the increase of the local computational power. These limitations mostly concern the network resources and battery power that are both still rather poor. Thus the main barriers for mobile nodes are network resources and limiting the power consumption of the device itself. Today, wireless networks provide limited reliability and less bandwidth than fixed networks. Moreover, all mobile nodes are highly energy dependent as they use batteries with a limited capacity. Additionally, roaming is a feature that increasingly must be supported for such wireless devices, as their physical portability leads to users to use them even as they move about. This may require the utilization of different wireless networks while the node is on the move. Therefore, for all the above reasons there is a demand for the development of intelligent mechanisms and techniques for optimizing the management of these limited resources, while exploiting the local resources, thus providing users with the best possible performance within the available resources.

At the present, there are operating systems, such as the Symbian OS [22], specially designed for supporting advanced features in mobile computing. However, there is still development to be done. Moreover, although there are many new applications for mobile computing, they are not yet sophisticated enough to cope with changes in the wireless environment, these changes occur due to the node's change in context. This creates a gap that should be filled by software between the applications and the operating system; this is frequently called middleware. This middleware provides a collaborative partnership between the operating system and the applications, assisting and making both more sophisticated, in terms of scheduling and managing traffic in a wireless environment. The focus of this project is how to utilize such middleware to best serve the needs of the mobile user.

Key words: Mobile devices, Wireless networks, middleware, context.

Sammanfattning

Mobila enheter som handdatorer och mobiltelefoner har kontinuerligt utökat sina användningsområden och popularitet. Samtidigt har de i och med det fallande priset blivit mer och mer attraktiva och tillgängliga för den allmänna marknaden. Detta har lett till utveckling av nya tillämpningar för sådana portabla elektriska enheter. Mobila enheter tenderar att få ökade lokala resurser som exempelvis större minne och CPU [2]. Fastän en förbättring av dessa hårdvaror har gjorts så karaktäriseras enheterna av begränsningar som inte har utvecklats i lika rask takt som de lokala resurserna. Dessa begränsningar handlar till större delen om nätverks resurser och tillförsel av energi via batteri, som båda fortfarande är relativt dåliga. De huvudsakliga barriärerna för de mobila noderna är alltså nätverks resurser och enheternas energikonsumtion. I dagens läge förser de trådlösa närverken en begränsad pålitlighet och lägre bandbredd än de fasta nätverken. Alla mobila noder är även väldigt energiberoende eftersom de använder sig av ett energibegränsat batteri. Förutom detta så är roaming en aspekt som måste öka för sådana trådlösa enheter eftersom deras transportabla egenskaper medför att användaren kan använda sig av enhetens funktioner även vid mobilitet. Detta leder till att det behövs tillgång till olika trådlösa nätverk i och med att enheten omplaceras. På grund av alla dessa orsaker som beskrivits ovan finns det en efterfrågan på utveckling av intelligenta mekanismer och tekniker för användningsoptimering av dessa begränsande faktorer, samtidigt som man maximerar de lokala resurserna och på detta sätt ger användaren bästa möjliga prestanda inom det tillgängliga området.

I dagens läge finns det operativ system, som Symbian OS [22], speciellt designade för att stödja avancerade drag inom mobile computing. Det finns däremot utrymme för utveckling av dessa och fastän det finns många nya tillämpningar för mobile computing så är de inte tillräckligt sofistikerade för att klara av ett smidigt byte mellan trådlösa nätverk när noderna omplaceras. Detta medför ett glapp som borde åtgärdas med mjukvara kallad mellanvara, ett program som förmedlar arbetsuppgifter mellan användarnas tillämpningar och datornätets resurser. Mellanvaran gör att operativ systemet samarbetar med användarens tillämpningar och gör att hela systemet blir mer sofistikerat i termer av trafik hantering i den trådlösa miljön. Tyngdpunkten i detta projekt ligger i hur man ska utnyttja sådan mellanvara för att optimera systemet i de mobila enheterna utifrån användarens behov.

Nyckelord: Mobil enhet, Trådlösa nätverk, mellanvara, sammanhang

Acknowledgements

First of all, I wish to thank Prof. Gerald Q. Maguire for his professionalism and for taking time to give feedback to my work. I would also like to thank my advisor Andreas Wennlund for being very helpful at the first steps of my work.

But most of all, I want to thank Karolina and the family Arandia for their great support and inspiration. Many thanks go to Massimo for his technical support and to Ruxandra for her help with the drawings.

Last but not least, I would like to thank all my friends in Sweden for spending amusing times together and for making my work more pleasant. Finally, I wish to thank my family for their patience of waiting for my return.

Table of Contents

List of Figures

List of Tables

Abbreviations

CDXP	Context Data Exchange Protocol
FTP	File Transfer Protocol
GPRS	General Packet Radio Switching
GPS	Global Positioning System
GSM	Global System for Mobile communication
IrDA	Infrared Data Association
KB/s	Kilobytes per Second
MANET	Mobile Ad Hoc Network
MC	Movement Circle
MMP	Mobile Motion Prediction
MT	Movement Track
PAN	personal area network
PDA	Personal Digital Assistant
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
QoS	Quality of Service
UMTS	Universal Mobile Telecommunications System
VHE	Virtual Home Environment
WAP	Wireless Access Protocol
WLAN	Wireless Local Area Network

1. INTRODUCTION

1.1 Example Scenario

A mobile node with multiple wireless network interfaces (GPRS/UMTS, WLAN, Bluetooth™, etc.) is moving within a heterogeneous wireless environment with multiple wireless cells with different ranges and characteristics. This node runs several applications, each one with different characteristics. Some applications require high bandwidth and low reliability; while others are error sensitive, but consume only a small fraction of the available bandwidth and capacity. Some applications are more time critical than others. Maximizing performance while trying to minimize the cost that the user will have to pay for the link utilization, is a central issue which must be addressed.

More specifically, imagine a user (Fred) with a handheld device with the above characteristics. The user is an employee of an international marketing company and he has to meet a customer at a restaurant in the center of the city. Before going to the appointment he has to take his little daughter to her kindergarten. After this he takes a taxi directly to the restaurant. While he is in the taxi, he uses his Personal Digital Assistant (PDA) to upload some photos he took to file server via FTP and to update his antivirus software (i.e. a download). At the same time he calls his wife to ask her what she wishes to have for dinner tonight.

Because his current location inside the taxi only gives him the ability to have network access via a GSM/GPRS link, he cannot use all the services he requested at the same time. Therefore, priority is given to the voice application, since it is a highly delay sensitive interactive service [7]. The GSM/GPRS link is utilized by the voice application, while the traffic for the other two applications is postponed until network conditions change and a better **opportunity** is found or until the call ends. The user of course is not directly involved in these decisions and can concentrate on his plans for today. Some time later, while Fred is still in the taxi, he finishes his call. However, he still doesn't have access to a network with higher throughput than that offered by GPRS. Hence, his applications for uploading the photos and downloading the new virus definitions are not triggered, since they are not time critical and they are expected to require quite a lot of bandwidth to be completed in a short time. As Fred does not need these requests to be completed urgently and because the GSM network is rather expensive for such a large volume of traffic, Fred saves some money by delaying these requests. Additionally, the unused network's resources can be used by other users, which is a benefit for the network operators and other users. Note that if there were no other users, then the operator might offer Fred services at a sufficiently low price that it would be worthwhile for Fred to use them.

Finally the user arrives at the restaurant, pays the taxi driver, and puts his handheld device in his pocket. As he enters the restaurant his PDA notices that WLAN network is available, as the restaurant offers free wireless access to their customers through a wireless hotspot. His pending requests are automatically triggered to use this WLAN for transmitting their data. While he is talking with the customer his PDA is working, utilizing the WLAN. At the end of the meeting when he returns to his office, he finds his photos uploaded to the web server and his antivirus software updated (see Figure 1.1).

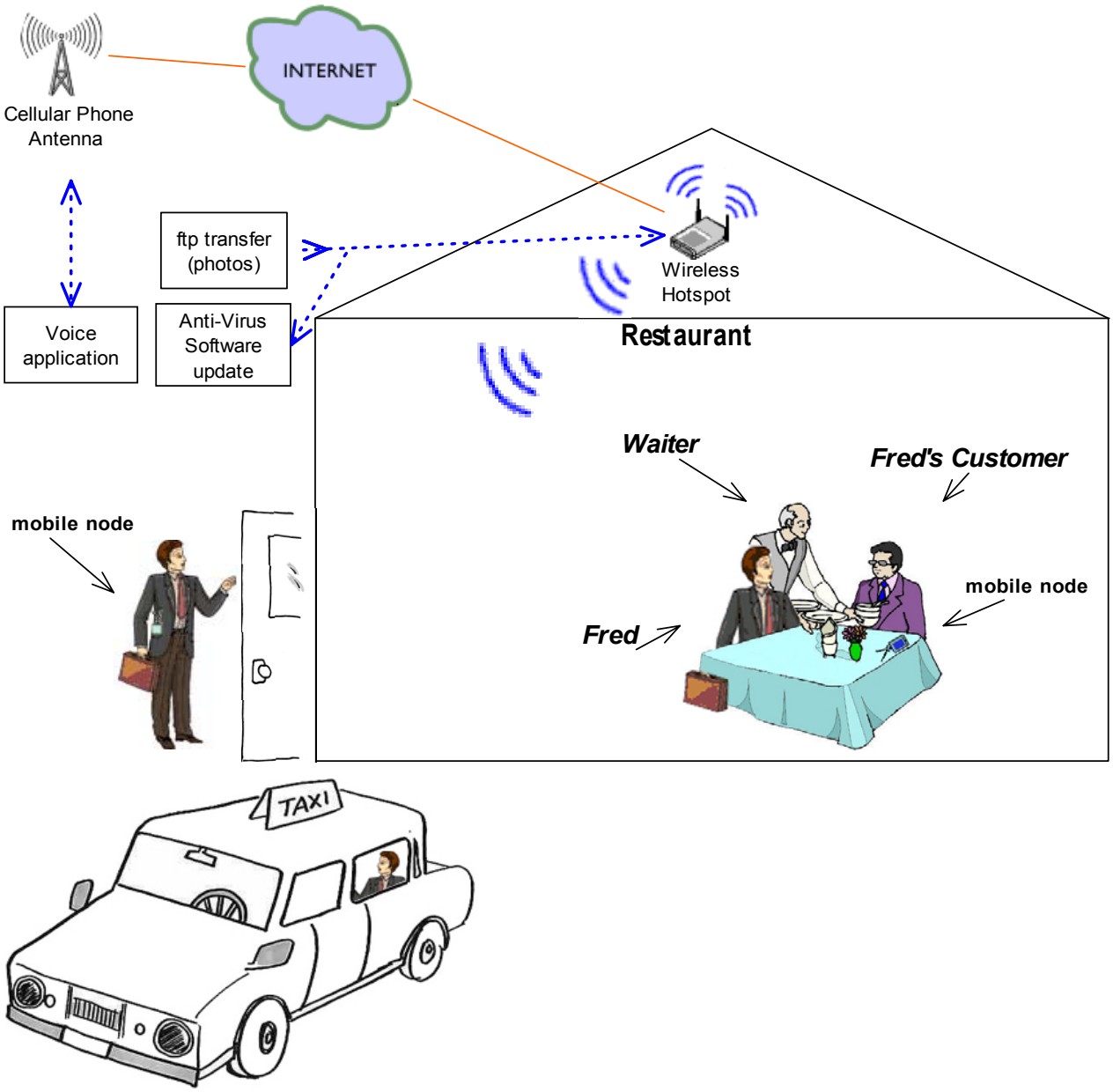

Figure 1.1: Fred's business meeting

In this scenario we try to maximize each application's performance while reducing interactions with the user. The user should feel that the services he requests are always available and he doesn't have to bother about **how** they are achieved and realized. We wish to support the user by providing him with the perception of having constant access to services. At the same time the middleware in the user device(s) considers how to minimize the cost of those services that the user requests. Thus, there is an effort to use cheaper links or at least avoid using expensive links when there is no urgent need for network access.

1.2 Current and Future Mobile Devices

Mobile devices, by nature, are devices designed to be portable. As a result their designers' primary considerations are weight, size, and ergonomics [3]. Power consumption is also a major aspect that must be considered, due to limitations in battery capacity. However, constructing compact and lightweight mobile device means that they have more limited computational capability compared to fixed PC's. On the other hand, mobile devices are equipped with attributes that the stationary ones don't have, such as multiple radio interfaces (GPRS/UMTS, WLAN, Bluetooth™, etc.) which have become very common in the latest portable devices.

Today's mobile devices are already multipurpose information appliances, often integrating camera, organizer, or music player in the same device. Popular devices such as Personal Digital Assistants (PDAs), cellular phones, and badges offer continuously improving hardware with modern user interfaces. Next-generation mobile devices are expected to have dramatic increases in storage capacity, but still limited battery capacity. For example, storage capacity is expected to reach the scale of Gigabytes to support rather fast processors, while at the same time integrating several multimedia abilities. The first cell phone with a hard drive of 1.5G capacity has already been introduced (Samsung SPH-V5400) [8] [10].

All the above innovations have made mobile devices more powerful and opened the field to more advanced applications that will exploit these hardware capabilities. Users expect to increase their welfare by exploiting the new services that portable devices will provide. Therefore network infrastructures have become an important concern, as some new services will either demand higher bandwidth and reliability from the wireless infrastructure or will need to adapt to lower bandwidth and limited connectivity. To realize this, intelligent utilization of available wireless network resources is required.

1.3 Problem statement

This research project aims to provide satisfactory services to end users of mobile nodes, i.e. as the user moves between different wireless networks, their mobile computing environment should support the services which the user wants in a way that provides user satisfaction for a reasonable price. Ad hoc connectivity may be available via other nodes and could be exploited. Moreover, mobile nodes may have multiple wireless interfaces that might be simultaneously available. Therefore the user's applications should be properly scheduled and operate via the optimal interface, so that the user experiences the best possible service. The user should not need to know about the current network conditions and the middleware should schedule transfers of the application data. This transparency aims to relieve users of the need to be constantly aware of the details of their computing environment, thus allowing them to focus on **their** own tasks.
We have a general goal of reducing the perception of the user that a service is unavailable or inadequate due to lack of network connectivity or excessive latency while using network

applications. This result builds upon the creation of policies that will consider network cost, when the service can be delivered, and what simultaneous services can co-exist.

1.3.1 A user's preferred experience

From the user's point of view, he or she should regard the services as being available at any location at all times. However, the user should not have to be concerned with the network details. Thus the user should not be concerned with technical details regarding available network links or required to make any decisions such as making choices about which network should be used for specific applications. Moreover, when network "problems" occur, such as no available links, weakly connected links (low-bandwidth or intermittent connectivity), or changes in bandwidth for current links, the user would like to experience unchanged QoS (Quality of Service). Such network problems can frequently occur as it is common in wireless channels to experience path loss, fading and interference.

Nevertheless, providing continuous and seamless services under all different wireless network conditions may **not** always best serve user's intentions. For example, the user may want to be aware of the cost of the link that he is using. This means that even though there might be some wireless link available, the user may **not** wish to use it. There might be cases when the user prefers to wait to use a cheaper link rather than use an available but expensive one. Or the user might want to use a connection with different characteristics from the existing one. Bandwidth and low link cost might not always be the criterion for user choices. For example, reliability might be important. Services such as internet banking or online credit card payment might not want to use a WLAN that has lower reliability as compared to GPRS [1], especially when there is little data to be transferred. However, an important requirement of mobile computing systems is the ability to access **critical** data regardless of the device's current location. For this case of critical data, ubiquity becomes a primary concern, i.e. when it is truly critical cost becomes secondary. This implies the use of user specified policies to provide user specific behaviour while avoiding or at least minimizing user interaction [23].

1.3.2 Application behavior

In order for the user to experience good service under different kinds of network conditions, the applications should be adaptive to changes in the network status and opportunistically utilize communication resources.

Adaptation in applications generally deals with situations characterized by variation in network quality and disparity in the availability of remote network resources. Concern is also given to battery power consumption, and low robustness resulting from the physical environment and motion.

Because wireless channels are subject to path loss, fading, and environmental interference, adaptation should occur for the applications' demands and the operating system which is handling the computing and communication resources. Adaptation is required when changes

occur in the network's characteristics. In order for the applications to continue working smoothly, they should dynamically conform to the limitations of their current environments. Therefore, avoiding bandwidth bottlenecks requires traffic shaping. For example, when we move between two heterogeneous networks with different characteristics or when the conditions change within the same network, an application may need to change its schedule for sending and receiving application data to and from the network. Some particular cases occur during handoffs between wireless networks or due to traffic reductions in link bandwidth because of weakening of signal or competing traffic of other users.

Nevertheless, adaptation in applications can be coupled with other resource management mechanisms such as opportunistic scheduling, in order to further increase network performance. Thus, another means of dealing with changes in the wireless network characteristics is to exploit this opportunistic behavior of applications. This generally means, exploiting opportunities to utilize network resources.

Exploiting these opportunities attempts to maximize network utilization, defined as the difference between the value of throughput and the cost of the operational link [4]. Another parameter that can be considered is minimizing transmission power, both to decrease energy consumption and to minimize impact upon others. This means that a high bandwidth network link does not always present a good opportunity and hence should be preferred to a lower bandwidth link, but rather that the offered traffic characteristics should match the link's resources.

How is this opportunistic communication achieved? What are the parameters chosen for identifying the best opportunity for sending data? Will this be configured by the user or will there be some constant, standard values that will apply in all the cases? Each of these questions will be investigated and suggestions will be proposed in the design and evaluation of the proposed middleware.

1.4 Middleware is needed

Applications by themselves cannot easily achieve this performance goal, as each application would need to be sophisticated enough to cope with opportunistic scheduling and adaptive behavior. This would make developing applications for the mobile nodes very difficult. Gathering information from the network and processing it locally would be required, and this procedure would be done *separately* for every application, resulting in increased load, increased bandwidth consumption, and increased local resource utilization.

On the other hand, using a single policy framework for all the applications seems like a better architecture and allows the applications to be simpler, hence such a framework is more suitable for mobile nodes. Therefore, middleware will be used to handle this complexity and provide applications with a single interface.

Middleware can be defined as a software layer between the operating system and the applications, which provides an interface between them as a common programming abstraction and serves some programming purposes (see figure 1.2).

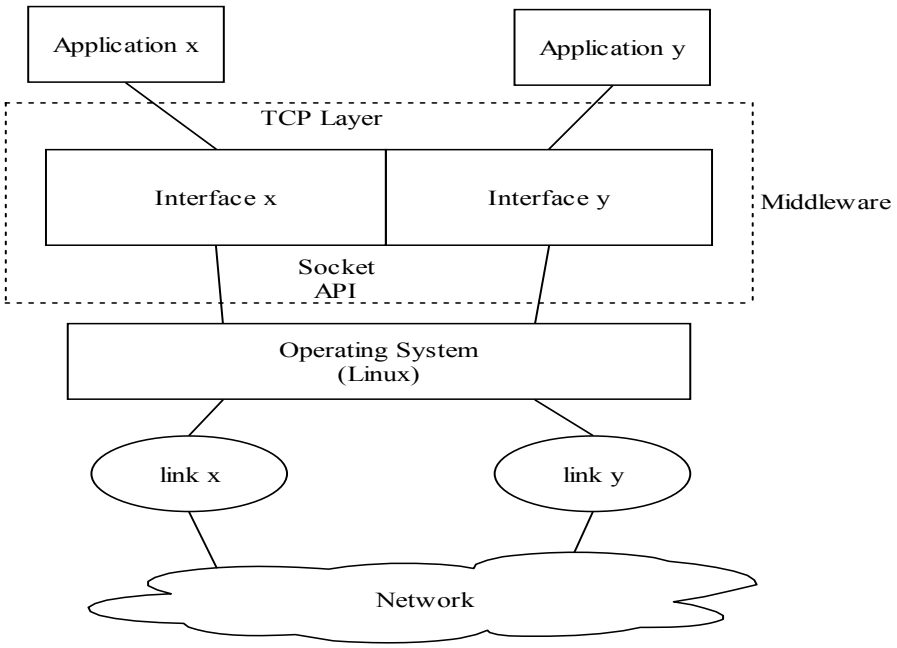

Figure 1.2: Middleware Representation

In this project a prototype of such middleware is designed and implemented on top of a Linux operating system. It interacts with applications at the transport layer. This is preferable since by having the interface in the transport level we can avoid TCP timeouts from reaching any of the applications. This middleware uses policies and context information as the major inputs for deciding upon the opportunistic behaviour of the applications. A description of potential context information is given in the next section.

1.5 Context Information

Traditionally, services are independent of their environment. However, when we deal with mobile devices, the environment is a major factor, as it continuously changes while the node is on the move and often influences decisions about what is relevant. Ordinary applications are strongly affected by these alterations in their context; therefore they should have an awareness of their context and how this information can be exploited to optimize an application's performance. Additionally, some mobile applications **want** to explicitly be aware of context, e.g. "restaurant finder", "friend finder".

In order to better understand the context's influence on applications, consider the following example. Imagine that you are working with your PDA which has an IrDA (Infrared Data Association) transceiver, WLAN, and GPRS/UMTS interfaces. Several applications are running, among them are a voice application and a task for printing a document. You are currently in Kista Galleria, where there is a WLAN hotspot to which you have access and you are currently using it for your Voice over IP based application. Your printing task cannot be executed as you don't currently have access to any printers. You start walking from the

shopping center to your office that is situated approximately 300m away. Near the middle of this journey you move out of range for the Galleria WLAN, but your conversation hasn't finished and you need to keep the voice application running. Therefore, a decision is automatically made to start using GPRS in order not to interrupt your conversation. As you approach your building, you enter your office's WLAN network; so your voice application switches back to using the WLAN interface. Moreover, the IrDA sensors inside your office sense the presence of your device through its activated IrDA interface and your printing request is triggered to use the local printer for printing your document. Eventually, you are finished using all the services that you have requested and have optimized your usage by exploiting context information and the capabilities that are locally provided.

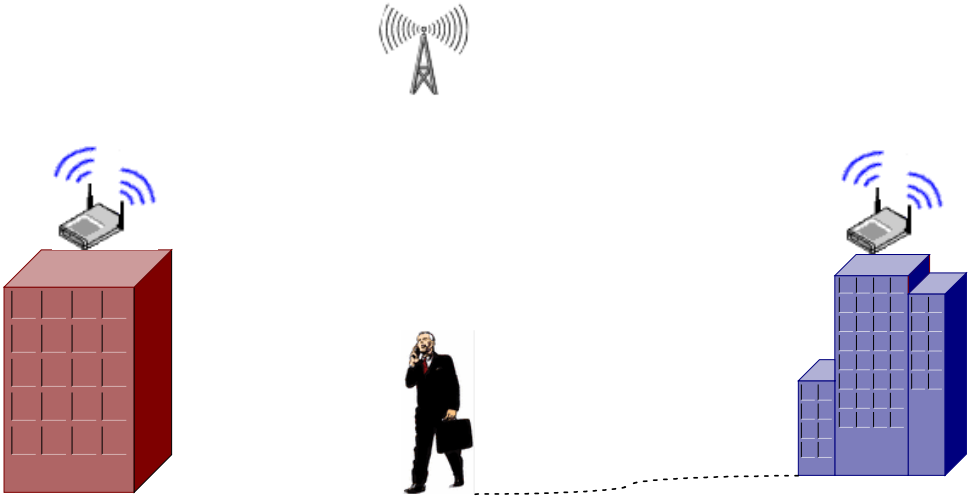

Figure 1.3: Walking Towards your Office

As context information we include *any* information that is relevant to the condition of a context entity and can be used to characterize this condition. Such context resource can be for example the temperature, light level, location, or the characteristics of a network link.

Embedded mobile devices with this ability to retrieve and utilize the context information are called Smart Devices [6] [29]. The utilization of context information is an interaction between the real and the digital world. As applications become context-sensitive, these devices can change their configuration according to their context and policies. Therefore, exploiting this context information is a key method to improve the performance of mobile services.

Context servers are used for gathering and distributing context information to mobile devices. These servers are usually stationary robust nodes that are connected to context sensors (thermometers, light sensors, etc). The information obtained by context servers provides knowledge of certain environmental parameters. The main purposes of these context-aware servers are to provide service allocation and discovery services [5]. They can notify mobile nodes about the services available in the current environment. This is particularly important for services available within a PAN (personal area network) or WLAN. Examples of such services include access to a local printer or to a file server.

Moreover, service delivery is also supported. This refers to the distribution of services that are available within a network and can be provided to neighboring networks. So, services derived from nearby networks are also available to the mobile user if there is a node which bridges between these networks. Distribution of context information and services to nearby networks and the other way round allows nodes to plan.

To collect context information, sensors are required. These can be physical sensors measuring values such as temperature and acceleration of a node. Additionally, software sensors operate as typical programs that track other software applications and report their availability. The results from these sensors may need to be interpreted to be meaningful to applications and the middleware.

But what is exactly the context information that will help applications' to exhibit opportunistic behavior? In our case we will be mostly concerned with context information regarding the *wireless network's characteristics*. Therefore, we are interested in reports regarding the availability of wireless networks at a given location and in other nearby locations as well. Some properties that should be taken into account are the available bandwidth, signal strength, as well as the cost of the link (per time or per unit of traffic). The current location of the mobile node and its predicted future location (according to earlier movement) are also of importance [24, 25].

Moreover, potential nearby ad hoc nodes should also be reported, as they might also be exploited for providing services. Predicted availability for currently utilized link and the status of the device (battery level, available memory …) are some of the additional parameters to be considered. All of the above context parameters are very significant along with their predicted future values.

The middleware should process all this context information and use this information as an input together with user preferences and user profile settings, in order to make decisions concerning scheduling and other policy based actions for the applications.

1.6 Opportunistic Communication

In what way will context information be used to enable applications to better utilize the network? Information regarding the environment will help applications to be opportunistic. Opportunistic communication is one type of adaptive communication. There are cases when the current context as specified by network conditions does **not** permit the applications to achieve their desired QoS or connectivity could even be totally unavailable. In order to address this problem, services may be postponed until a better opportunity occurs. A good opportunity may generally be characterized by propitious characteristics of an available wireless link, such as low cost combined with low delay, high throughput, and high reliability [7]. Opportunistic communication may also be planed in advance, in a *proactive* way. There might not be requests for available services at a certain time, but it is speculated that there will be requests at some time in the future. However, the current network conditions (low traffic, high bandwidth, low cost, …) may encourage the utilization of an available link. Therefore, action can be taken to pre-fetch or pre-deliver *anticipated* application data. For

instance, update or backup processes can take place at non-critical times when there is no competition with other applications. Another case is when a service is predicted to no longer be available and therefore the opportunity should be exploited to immediately access this service and cache data **now** rather than waiting.

Opportunistic communication can also be coupled with other resource management mechanisms to enable better network and local resource utilization. Opportunism not only exploits the variation in channel conditions, but power-allocation and decreasing the power consumption of local resources should be taken into account in scheduling communication operations [4]. Limited resources such as the battery capacity play a critical role in this management policy, as these limits might not allow the completion of the data transfer for a service, hence starting to use this service might be delayed. Nevertheless, a drawback of opportunistic scheduling will be increased delay in delivering some services when they wait for the appropriate opportunity. Thus, a good management policy that calculates the tradeoffs and allows optimal service delivering at the proper time is necessary.

2. PREVIOUS AND RELATED WORK

2.1 Related work

2.1.1 Relevant Context-Aware Techniques

There are several techniques that enable the utilization of the context information in order to increase performance. These relevant methods constitute complementary or contributing projects to the work of this project. Some of these are mentioned below.

2.1.1.1 Handoffs between GPRS and Wireless LAN

Mola and Inho et al. have shown that handoff policy may be applied to wireless networks [1] [9]. Their goals are the same as ours; i.e., facilitating mobile computing. Intelligent handoff policies attempt to prevent application interruption due to a broken link by exploiting wireless diversity. This means that the mobile node switches between different base stations potentially in different networks so as to provide continuous and ubiquitous connectivity to its applications.

The major advantage of this method is that you continue to have connectivity, thus your applications are not interrupted. This means that programs don't need to be restarted in order to reestablish connections.

However, you have to use Mobile IP [9], which is a routing solution for IP networks enabling handoffs to occur while maintaining a constant IP address. This is necessary because when you move between different networks your node is assigned a new IP address that belongs to the current sub-network. Without Mobile IP your running applications would be affected by this change. This happens because the servers that your applications are interacting with know the previous address of your node, specifically the address it had when the connection was established. However, applications should be informed of this change in address so as to avoid connection failures. For this reason a node (the home agent) must exist that it knows the current IP address of the node; this node is always informed after the mobile node changes IP address. So requests are transmitted to the home agent about the mobile node's new IP address, and all traffic is readdressed to the mobile node. After these requests arrive at the home agent, they are processed and the Mobile IP address of the node is translated to the IP address that the node was assigned in its current subnet.

However, the propagation of the mobile's current address causes extra delay. Applications may experience some additional delay during the handoff procedure. This is because there is a gap between the times when the current network interface is no longer available and when a new connection is established that can be utilized by the application's traffic.

Additionally you might have unnecessary handoffs. This can occur when it is predicted that your current link is weakening in signal strength and will soon be unavailable. If the only other link you can switch to is not much better than the previous one; then there is a need to avoid an immediate hand off back to the previous network. The solution to this is a policy which has hysteresis.

2.1.1.2 Multi-hop wireless Networks

Mobile computing can utilize multi-hop communication techniques. The main goal is to reduce the total cost of the network. The wireless infrastructure cost can be reduced with the use of Multi-hop Capable Nodes [11]. These nodes offer ad hoc connectivity and expand the coverage area of a Base Station. The idea is to replace access points and base stations by multi-hop capable nodes, in a way that optimizes the overall performance and minimizes the cost for the service provider.

The benefits of this approach are the extension of the coverage area for a wireless network while decreasing the operational cost for delivering services to the mobile users, by using cheaper intermediate propagation means. However, these techniques require an increase in the mobile nodes' memory, as they will have to carry more messages and propagate them to other nodes. Furthermore, the delay will increase in service delivery and battery power use increases. This increase in delay occurs because of the node to node propagation of the traffic in order to reach the final destination node [46]. Therefore, this model better suits delay *insensitive* applications such as email, messaging, and file sharing.

2.1.1.3 Location Prediction Algorithms

Context information is closely related to location, as for every location there are different conditions that provide us with context parameters. Therefore, change of location usually brings also change in context information. Predicting future context parameters for a mobile node requires the prediction of its future location. There are algorithms that are used to predict the future location of a mobile user [24] [47]. Most of these algorithms are based on humans' regularity of moving in the same places. This regularity can be hourly, daily, or weekly. These algorithms that take as input the user's movement history patterns are also called MMP (Mobile Motion Prediction) algorithms [48]. There is always a regular and a random part in user's movement. The regular part can be examined and predicted with MC (Movement Circle) methods, for longer term regularity based on the assumption that the user returns to the same point he started and MT (Movement Track) methods that is the less constrained form of MC, that consist of several MCs [24]. The random part can be evaluated with probabilistic analysis using stochastic processes such as Markov Models [47]. Given that we know the current location of the mobile user and also have the output from our movement prediction algorithms we can predict his future location. This enables us to be prepared in the user's future location, by pre-assigning and pre-connecting services at that location. Some of the procedures that can be followed are the prediction of the future state of network topology, the proactive route reconstruction, the improvement of the routing protocol performance, the resource allocation and the bandwidth reservation.

2.1.1.4 Context Servers – Context Databases

In order for mobile computing to utilize context data and to become context-aware requires exploiting the data that are acquired by the context sensors (thermometers, network detectors, etc). However, this data is not necessarily formulated in a way that can be used by higher level context information management modules. An intermediate party supporting the manipulation and formulation of the data collected by the sensors is a Context Server [5] [28]. A Context Server is generally a robust fixed node that is linked to a database. This database is used as a repository for storing context information regarding a particular location/entity/… that the Server is monitoring. The Context Server acts as a "local" server that is responsible for information associated with its local environment.

A Context Server's main operations are the collection, formulation, distribution, and access control of context information. The procedures for collection and formulation were described above. The distribution and access control of the context information for mobile clients are key features for the server. The access to the context information that is being stored in the server's database is controlled by policies that define the different access permissions for each mobile client or subscriber. This means that private data policies and user privileges must be considered.

The necessity for a Context Server is apparent, as it provides an abstraction and simplifies the utilization of the context information. However, there are some drawbacks to this technique. A high level of security is required in order to protect the privacy and critical data of the users from other "malicious" users. For example, context information that is stored in server's database might include a user's current location, services that the user is using, and a set of locations that the user regularly visits. This kind of information should be treated in a secure way and not be distributed other than as the user wishes. Moreover, latency or updating problems of the context information might occur. The server should have the latest context data and be able to quickly deliver this information when asked by a client. However, if that the server has many subscribers to serve or the values of the context entities that it monitors have changed without the database being updated, then clients may experience latency in the information dissemination or receive of inaccurate data [53].

2.1.2 Related Projects

There are also some related projects in this area that involve several methods for providing context services in a unique service delivery. Some of these projects are described below.

2.1.2.1 William Schilit's Ph.D. thesis

William Schilit describes an architecture for supporting context-aware mobile computing in all its stages [14]. The design of this system integrates procedures for exploring the computational environment, defining the information needed by the applications, addressing the discovery of this information as needed and after this exploiting the collected context information while efficiently distributing it to nearby context-aware mobile environments.

Techniques such as traffic scheduling and traffic shaping are provided to make applications adaptive to network changes. Therefore application data can be pre-fetched and applications requiring high bandwidth can be reduced to lower bandwidth providing decreased performance but without interruption. For example, a high quality video may reduce its resolution when the network conditions change but it can keep playing.

Emphasis is also given to further context aspects/entities such as people who are nearby and the nature of the location that the mobile node is. For example, if the person using the mobile device is in a conference room attending a meeting, his mobile node should automatically enter a mode that will allow silent operation, i.e., without audio notifications of incoming calls and messages. When the user is in close proximity with the party that they wish to talk to, the mobile device should notify/remind him that there is an opportunity to directly contact this particular party. Moreover, the user is given the ability to keep track of both nearby and remote context entities so that he can navigate to a desired direction. A record is also kept of located-objects and persons that the user has encountered. Thus, the user can discover the context information that best fits his needs.

2.1.2.2 Cello Project

The Cello ("Cellular Network Optimisation based on Mobile Location") project mainly concentrates on optimizing the utilization of 2G and 3G cellular networks [15]. The main purpose is to utilize the context information that the mobile nodes provide, by exploiting the mobile location technologies that are offered. Location capable phones with an integrated network monitoring system are used as terminal nodes for this project. They act as sensors for measuring context information and informing context servers; in this case a Mobile Network Geographic Information System (MGIS). This server collects location-related data such as measurement-based coverage areas and location-specific handover performance data. All this location-based data is obtained from the mobile terminals and is used to improve the mobility management functions of the network.

Some actions that are performed by the system are the detection of problematic areas in the event of network overload, creation of alternative network plans for this situation, and switching to alternative base stations (handoffs) when appropriate. The goal of this project is to enable a 2G or 3G cellular network to optimize its own performance. This can be done by cooperating with other networks, in terms of (spatially) distributing users and using the context information that they posses. Consequently, the network's capacity is increased, resulting in the ability to serve more potential service requests from users.

2.1.2.3 Aura Project

The Aura project's main aim appears to be the establishment of ideal working conditions for the human-user [16]. It attempts to serve a user's needs for high productivity and to be distraction-free. In order for this to be achieved, context-aware intelligent services are provided. They have created a framework for supporting future artificially intelligent services that also require context information. Therefore, new techniques are being developed for exploiting context information in a way convenient for the applications.

Particular effort is being made to cache and pre-fetch data that is predicted to be useful for the user, in order to relieve network resources at critical times of overload. Several different algorithms were developed for predicting possible requests and their associated traffic. These assumptions of the potentially useful traffic are based on *past events*.

Additionally, new techniques have been developed for location tracking. As GPS equipment would make mobile nodes less portable, in terms of increased weight and power consumption; new algorithms were employed for location sensing. Location is determined by measuring the signal strength from a mobile node to all wireless access points. The comparison of all these measurements to a table containing signal strengths for each location, gives an approximation of the user's location.

Another important issue being focused on in this project is the determination of the user's intent. This refers to the user's preferences regarding his expectations from the services provided, in case of inconvenience caused by a resources' unavailability (for example, due to network failure). It is important to know what the user wants to do with in a running application; when the application is not able to perform as usual because of the lack of connectivity. For example, we should consider the user's preferred privacy policies concerning services, the user's plans for moving from one location to another, and the resource requirements for the user's future computing activities

2.1.2.4 VESPER project

VESPER project aims in enabling the VHE (Virtual Home Environment) concept [49]. VESPER is a European IST project with research labs, universities and telecommunication operators from the industry as partners [32]. Its goals are to support roaming users with service probability, session mobility and service scalability. This can be achieved with

system architecture that will be designed to operate in an environment with heterogeneous networks and multiple providers. The project involves personalisation of the service environment which allows to the user to set his service preferences by editing the user profile, portable services that are accessible from everywhere, mobility issues that decide upon suspending/resuming or roaming (keep running) a service. VHE is a new aspect that attempts to provide with a single common interface for the service developers.

2.1.2.4 CONTEXT project

The CONTEXT project attempts to provide complete end to end context-aware service provisioning for the users [21]. Middleware is introduced for dealing with the discovery, acquisition, distribution, and management of the context information. By adding context functionality in the network infrastructure, this middleware aims to customize all different kinds of context-aware applications and to provide integration of multiple types of context entities [12]. Unfortunately there is no further information available.

2.1.2.5 Owl Context-aware System

Owl is another system providing context-awareness to services so that pervasive computing is achieved, while the user's attention to communication details is avoided [19]. Owl's approach to context information can be either transient or persistent. The transient information contains data obtained by sampling the environment at a particular moment, whereas the persistent case refers to accumulation of transient information that was acquired during certain past events.

People wish to move in unfamiliar territories, and expect to experience services that are related to their current location. However, privacy matters arise in the dissemination of user's context data (particularly the user's location and service accessed). Hence, privacy policy is also a very important issue, when retrieving context information and when distributing this information. Therefore, a role-based access control mechanism is employed to deal with privacy policy management. This enables only authorized context management systems to access the context information transmitted by mobile nodes.

Inferring a user's intention enables the system better adapt to the user's needs. This means that actions are taken to anticipate useful information for the user in the future by pre-fetching it; thus providing better performance **and** availability of services while avoiding distracting the user.

OWL also discusses some advanced issues concerning the quality of the context information and providing a classification for it. Scalability and extensibility of the system appears to be a future goal, as it is expected to support 10 million diverse context information entities and serve approximately 1 million clients. The extensibility part seems to be an easier challenge as context-aware computing is comparatively new area and that future services can be *designed to comply with it*.

2.1.2.6 Kimura

Kimura is a system that attempts to optimize the efficiency of traditional office tasks by expanding the office computing environment [17]. This is achieved by using additional peripheral visual displays and introducing multiple interfaces for human computer interaction. These displays are used to present physical and virtual context information to contribute to the user's knowledge and understanding. The user can see the hypermedia relationships between his documents and this context information.

2.1.2.7 Solar System prototype

This system creates an abstraction for context-aware applications, which enables them to access context information [18]. The system operates by collecting, aggregating, and disseminating context information to subscribed applications. It is a delivery mechanism for context-aware pervasive-computing applications and it is designed with a graphic interface, integrating operators which can transfer the data as it flows in the graph.

Operators in the graph are used to manipulate the context information. There are several kinds of operators, including: filters, transformers, mergers, and aggregators. Filters operate by selecting and delivering a subset of information. Transformers transform the context information from its present format to a format that can be exploited by the applications. Whereas, mergers simply send the **set** of events that are received and aggregators integrate different context information into a single entity.

It was observed that context-aware applications usually subscribe to similar context data. Thus operators can be re-used in different settings and correspond to different application requests. Therefore they form an operator graph. This graph extracts context information derived from the operators; this information is distributed to the network. The aim is to minimize the traffic across the network due to transfer of context information, while allowing the computation to be distributed.

2.2 Evaluation of Previous Work

Before proposing and designing new techniques we should be consider previous related work. The evaluation of this work will help us draw some conclusions about the parts that still need to be developed.

The prior work regarding *Handoffs between GPRS and WLAN networks* has contributed to ubiquitous computing by facilitating the seamless utilization of different wireless networks without connectivity interruption. This utilization takes place by exploiting the context information in terms of available network resources to facilitate better application performance. The main concern is to provide continuous connectivity but always choosing the networks that *will provide* the applications with the best possible performance.

Multi-hop wireless communication between mobile nodes may provide us with the ability to establish ad hoc connectivity between different mobile nodes. In this case the context information enables exploiting available nearby mobile nodes and their attributes that can make the suitable for establishing connections that can eventually link to a backbone network.

Moreover, the existence of *Context Servers* and *Context Databases* makes the exploitation of the context data easier for the mobile devices.

Some projects such as *Cello* equipped mobile nodes with the ability to perform as sensors collecting information regarding the current 2G/3G cellular network where they are; this information was then exploited to improve of the network performance. While other projects such as the *Owl Context-aware System*, focused on privacy and security aspects that should be considered when context information is being distributed. This mainly refers to context information regarding private user data.

Furthermore techniques have been developed for collecting, aggregating, and disseminating context information, some of these techniques were analyzed in the *Solar System* project.

Judging from the prior related work there has been significant development in the area of collecting, formulating and distributing context information. Advanced work has been done to exploit context information in order to provide services such as ubiquitous connectivity and computing. Context information also enables techniques that server the seamless connectivity even when the mobile node is on the move. However, the area of service provisioning appears to be less developed in terms of utilizing the context information for improving the performance of applications by using **scheduling** techniques. Such techniques can result in better utilization of the network resources and this can alleviate the network overloading. This approach will be more extensively analyzed in the next chapter.

3. MIDDLEWARE DESIGN

3.1 Introduction to Context-aware Middleware

In general, the goals of the proposed middleware are to provide management and scheduling services for applications that run on mobile nodes. This middleware should be designed to interact with the applications by granting or denying network access or scheduling future reservation of network resources for a particular application. This system can be described as in abstract black box that takes as input context information that concerns the applications, application requests for accessing the network, and optionally some configuration from the user. As output, it produces a schedule that will enable the applications to efficiently utilization the network's resources. This abstract view of the middleware's functions can be seen in the following flow diagram (*Figure 3.1*).

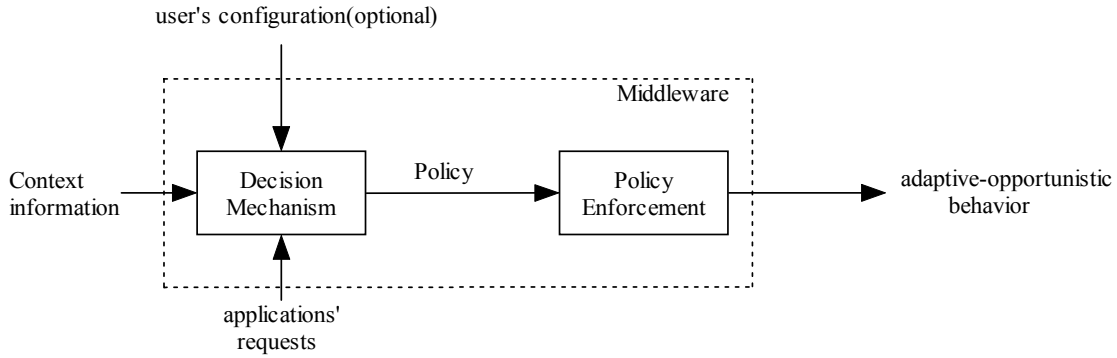

Figure 3.1: Middleware Abstraction

3.2 General Middleware Architecture

3.2.1 Overview

In the following design we will attempt to describe how middleware for context exploitation generally operates. There are numerous procedures that can be applied based on the utilization of the context information, such as different policies for application scheduling, data buffering while there are insufficient network resources or data pre-fetching. It is impossible to design and analyze in depth all of them within this report. Therefore we provide a general overview that will help to understand some general principles. Following this is a more detailed design of a part of the middleware regarding data "pre-fetching" and a demo implementation of it.

Figure 3.2 shows a more detailed view of the middleware's architectural design. Context information is delivered to the mobile node by a context server in a specified format [28]

[26]. Applications subscribe to the middleware, requesting access to the network. Decisions are made and each application is separately scheduled.

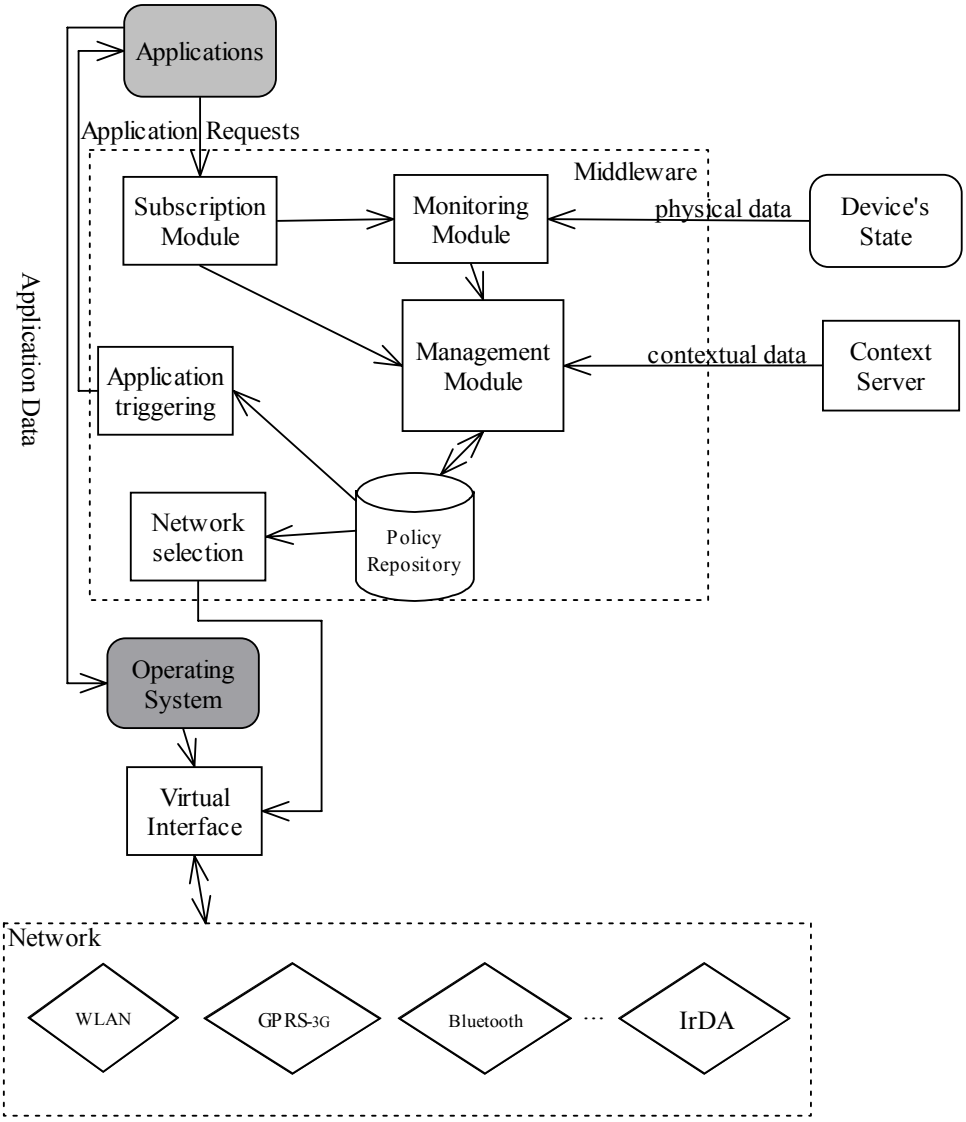

Figure 3.2: Middleware Architecture

IP is the common protocol used by all the mobile nodes. This makes the design and the implementation simpler.

3.2.2 Context Information

In our design we will mostly deal with context entities that refer to the different wireless network interfaces that these mobile nodes may have access to. As can be seen in Figure 3.2 the context information that is collected concerns the interface and is used to select the network: WLAN, GPRS-3G, IrDA, etc. The characteristics of these heterogeneous wireless

networks, such as availability, cost, and bandwidth, will be monitored and analysed in order to have an application efficiently utilize of these networks and their resources. This information is provided to the mobile node by context servers, which are nodes responsible for accumulating and distributing context information to context-aware mobile nodes.

3.2.3 Context Server

The *Context Server* for our system will be the context information provider to the main module of our design, the *Management Module*. We will not go into detail about the operation of the context server, as this is out of the scope of this thesis. A general description of the Context Server is presented in section 2.1.1.4; for more details refer to [28], [5], and [26]. However, it is important to mention that the context information is received by the mobile node in a predefined format and it doesn't need any extra processing to be used by the mobile node's context management tools.

3.2.3 Applications – Subscription Module

Applications that require the utilization of network resources have to be scheduled according to context and other parameters. Therefore the applications will not have immediate access to the operating system, due to its limited ability to perform advanced scheduling. Middleware will be used as an intermediary to allow subscriptions from applications, so that the middleware can provide scheduling of requests. Then, when appropriate the middleware will trigger applications to start transmitting data. This will implement the *policy enforcement* part of the middleware by interacting with the operating system and redirecting application requests to it.

Applications provide services and should be characterized by some parameters so that their service delivery matches the QoS standards of the particular service. From a user's perspective services can be classified into different categories according to some characteristics that are of critical importance for the services. Therefore, a suggested classification can be based on attributes such as throughput, delay, delay variation (jitter), interaction, reliability, or security [7].

Throughput is defined as the data transfer rate for the application data that are transmitted via the network, classifies the applications to those who require high throughput such as video streaming or others for whom throughput is not of major importance such as e-mail or messaging services. This is a very general classification as services may change their throughput requirements over time and an average throughput is required so that the QoS parameters of the services are met. For example, an e-mail service doesn't require a high throughput to be delivered continuously. But in the case that the device's unused bandwidth resources are utilized by other higher priority applications, the e-mail application's access to the network is delayed. This decreases the average throughput of the e-mail service.

The delay attribute refers to the deadline that its service has to meet for its delivery. It is also closely connected to delay variation or jitter. Some services are delay sensitive on the scale of

milliseconds or seconds, such as video or audio conferencing while others allow minutes or hours for completion, e.g. e-mail or file transfer.

The interaction issue for a service is closely related and defined by the allowed delay for that service. Thus real time communication like instant messaging or telephony should have low delay.

The reliability *pattern* refers to the pattern of errors in the communication. Services can have low or no error tolerance, therefore error correction protocols should be used to ensure error free transmission (e.g., services such as file transfer or banking services). In the case of error insensitive services such as video or telephony special concern need not be given.

Security also characterizes the nature of a service. Highly security sensitive services such as banking services or other services sending private data should be treated in a secure way via the use of the appropriate security protocols.

Nevertheless, from another point of view services can generally be categorized according to their required bit rate and completion time. *Table 1* shows these different categories.

	Service type	Application Example
Real Time	Constant Bit Rate	Videoconferencing, telephony
	Real Time Variable Bit Rate	Compressed audio/video
Non Real Time	Non-Real-Time Variable Bit Rate	Critical Data
	Available Bit Rate	LAN interconnection
	Unspecified Bit Rate	File Transfer, messaging

Table 1: Service Classification

Real time *Constant Bit Rate* services require a fixed amount of bandwidth and a specific Quality of Service in terms of delay, jitter, and cell loss. However, *Real Time Variable Bit Rate* services share the same characteristics with the difference that the traffic that is created by the applications is rather “bursty”.

In *Non-Real-Time Variable Bit Rate* services applications are non-real time with “bursty” traffic and they are error sensitive, but **not delay** sensitive. *Available Bit Rate* services are comparatively undemanding and can adapt to the available resources that remain from other types of services. Finally, *Unspecified Bit Rate* services are the most flexible in terms of requiring little service guarantees.

In our design we will treat applications as entities that have several attributes that will help us decide upon the opportunistic behaviour of each application and create a schedule. *Table 2* illustrates the relation between an application and attributes. Most of these attributes were explained above. The *delay* attribute will be used later to describe how long an application will be delayed waiting for other applications to be finished.

Interaction
Throughput
Delay
Reliability
Security
Delayed

Table 2: Application attributes

In the programming phase of the middleware these attributes (interaction, throughput, etc.) could be represented by integers that indicate the importance that they have for the particular application. These values would be compared across application so that policies can be created and enforced for every type of application.

As it can be seen in Figure 3.2 applications communicate with the middleware through the *Subscription Module*. The data flow that applications send, is requests for hardware resources such as network access that applications would normally send to the operating system. The *Subscription Module* is responsible to taking these requests and communication these application's subscriptions to the middleware. This procedure mainly involves assigning of values to the application attributes (as defined by our design, see *Table 2*). In order to perform this operation the *Subscription Module* should be capable of recognizing the type of application and classifying it by assigning values to the attributes. For example, if the application is a file transfer application, the attributes could be interaction=0; throughput=0; delay=0; reliability=1; security=0; delayed=0. "1" indicates sensitivity of the application to this attribute, while "0" means the opposite. For simplicity attributes are shown with "boolean" values, which means that they can either consider this attribute or not. However, there can be different scales with more levels for the values that indicate the degree of sensitivity for this attribute.

A pseudo code example for the procedures of the S*ubscription Module* might be the following:

```
/*applications can have one or  more of these attributes*/
app x{
int interaction;
int throughput;
int delay;
int reliability;
int security
int delayed;      /*this will be used later when the application request will
          be delayed in the stack after other applications take its place*/
}

request_subscr(x){

array app_array[]; //array for buffering the application requests
```

```
/* some examples of applications */
if (app==ftp){
        interaction=0; throughput=0; delay=0; reliability=1; delayed =0;
}

if (app==video){
        interaction=0; throughput=1; delay=1; reliability=0; delayed =0;
}

if (app==msn){
        interaction=1; throughput=0; delay=1; reliability=0; delayed =0;
}
        .
        .
        .

app_array[last++]=x; //at the end the application request is buffered in an array;
}
```

The app_array[] remembers the priority which the applications have for utilizing the network. Later the management module will re-order the application requests in the array, according to the application's attributes and other context data.

Middleware through its subscription module should be able to identify the type of application that subscribes to it. This identification can be done by examining the packets' headers that the applications send to the network. But in order to know the amount of traffic that an application will require, we have to examine application's first packet that establishes the connection.

For example, in the case of an FTP transfer, in order to know the size of the data to be transferred, we first have to establish the connection with the FTP server. However, this might be a problem for our opportunistic behaviour. Imagine the case when the data to be transferred has a big size that does not allow it to be transferred immediately but it is better to schedule such a transfer for the future, when the network conditions will permit it. But, the connection with the FTP server is established and the server reserves resources for a client that does not use them.

The idealistic communication in the previous scenario would be to know the size of the data without establishing a connection. But the FTP client works in a way that requires first the establishment of the connection and then it can specify the transfer that wants to do. A possible solution could be to exclude this kind of application from subscribing to middleware. Another solution for this particular case could be to try to simulate a "fake" FTP server that will respond to the client and establish a connection with it so that we learn the traffic that needs to transfer and schedule it by re-establishing a real future connection.

3.2.4 Device State – Monitoring Module

Consideration should be also given to the internal characteristics of the mobile node. Middleware should consider network resource requirements as well as local resource requirements, such as CPU, memory, or battery. The aim is the delivery of a service that will not be interrupted because of inadequate of local resources. For example, scheduling of an application should consider the total resources needed for the application to terminate, considering the local resources. The battery's lifetime will determine whether this application can finish within the current available power and whether it is advisable to start its execution. For example, we might not start a file transfer application of many megabytes but instead choose to run an e-mail client sending an e-mail of only a few bytes. This choice can be extremely critical when the battery's remaining capacity is low. Therefore, the middleware provides a feature (*Monitoring Module*) that monitors the mobile node's available resources and estimates the time needed for a service request to be completed.

The data acquired by the mobile device regarding its local resources can be also treated as context data. For the middleware this is the same type of input as other context data. However, to better represent of the actual situation we use a different definition for the data that is collected from the mobile node. Middleware monitors the device's local resources such as CPU, memory, and battery with its *Monitoring Module* (*Table 3*). This happens by periodically capturing the current values for these resources. For example, we are interested in the available CPU capacity, the available memory (in MB), and the remaining battery power. These values will help us calculate certain parameters and to estimate the behavior of a particular application under these conditions. As a result we aim to estimate the time that will be needed from an application to finish assuming that it starts to be executed at this particular moment having these available local resources. Then, there can be a comparison between the resources that the application requires to be finished and the operation resources that were provided to the mobile device given the remaining battery power. This information is passed to the *management module* that is responsible for taking decisions.

Battery
CPU
Memory

Table 3: Device Data

3.2.5 Management Module – Policy Repository

The *Management Module* of *Figure 3.2* is the core of our design. It is responsible for the decisions made when scheduling the application requests. It encapsulates the decision-making mechanism that evaluates all the different parameters of context and non-context information and applies policies that are been applied to the different classes of the applications. These policies are also stored in a policy database (*the Policy Repository)* so that the next time a similar application subscription-request occurs it cause processed quickly.

We must consider what scheduling decisions should be made by the middleware, through its *Management Module*. Once applications have passed their requests for utilizing the network to the middleware, they now wait before transferring application data. The options for an application are either to wait or to be triggered to transmit its application data. How will this application data be handled by the middleware? Several methods exist to assist applications increasing their performance in terms of efficient network utilization. However, these methods can affect other mobile local resources (such as disk capacity, CPU or memory usage). Therefore it appears that a tradeoff is necessary, in order to select the optimal method.

One method for optimizing the application's access to the network could be to **buffer** the application data. Buffering can take place mainly when the application data cannot be transmitted via the network due to the lack or inadequacy of an available link. Local buffers are used, utilizing the mobile node's local disk or memory capacity. After the *Management Module* decides to buffer some application data, confirmations are send back to the applications and they are assured that their data has been transmitted. When having big delays this method can mean the violation of the end-to-end semantics for the applications and it is better to be avoided for real time interactive applications. Later, the middleware prioritizes and sends the data that it has buffered from all applications. This means the choice of the appropriate network interface and the appropriate time to send the application data is managed by this module. After the data is sent to the network and is **acknowledged,** the local copy in the buffer can be deleted. The sent data may be buffered by some other ad hoc nodes in the same way as in the source node. After the data leaves a node and it is acknowledged, the node assumes that it will reach its destination and the transmission becomes the sole responsibility of the next node. This method helps in the efficient memory usage.

The main positive feature of this buffering method is that it maximizes the user's experience of a service. The user can experience nearly ubiquitous network connectivity and regard a service as available always and everywhere. A result of this buffering is that the user doesn't get any messages regarding the network being unavailable. She simply runs an application and she assumes that it will be completed, without needing to know any details about when and how. However, this buffering policy can apply to non-interactive applications **only**.

Nevertheless, there are some weak points that arise from this method. The first is the scalability that can be supported by the implementation. What if the user starts running many applications that require high bandwidth while there are no or insufficient network resources? How big should the buffers be and what happens after a buffer is full? Will the user eventually experience a delayed lack of connectivity? Moreover, although the user thinks that there is ubiquitous connectivity and the applications data is sent, it doesn't actually happen at that time. Applications are also unaware of this reality. Buffers utilize local resources resulting in a reduced performance of the mobile node. Nevertheless, the main disadvantage of this method is the delay in the provision of the service. This delay can consist of other delay components such as processing, queuing, transmission, and propagation delay [46]. Processing delay refers to the time needed by the packet to be assigned to an outgoing link queue for transmission at its source node. Queuing delay is the period between a packet is assigned to a queue for transmission till the time that it starts being transmitted. Transmission delay is the time that passes from the transmission of the first bit of a packet till the transmission of the last bit of the same packet; and finally propagation delay is the time

between the transmission of the last bit of a packet at the source node and the reception of that last bit at the destination node.

Another suggested method involves **pre-fetching** data from the network that is highly likely to be requested by the user's applications. This data can be chosen using algorithms for traffic prediction that we will describe in section 3.3.4. The pre-fetching can generally be performed when we have an idle network and a high bandwidth link. Thus when the user (through an application) requests data from the network, there will be a local lookup in the node's buffer to see if this data already exists. If the data is not locally cached, then we will use the network. Data can also be cached in the mobile node after being retrieved and used by an application.

The advantages of this second method is that there is a very efficient use of the network resources, since there is an attempt to minimize its utilization while there is a high load, and also maximizing the network utilization when it is idle, by pre-fetching data. The result is shift the utilization of the link bandwidth usage, to a more even utilization. Nevertheless, a disadvantage of this method can occur when there is no connectivity or the connectivity is lost and the needed data is **not** cached. The user will then experience an interruption in her service. It would have been better if this service began executing some time in the future when the conditions were more optimal. A drawback for network providers may occur because pre-fetching maximizes the network usage, although it might be *unnecessary* utilization. Therefore the network load is significantly increased in the backbone network and this may have a negative impact for the network provider as now fewer resources are available for other clients.

As a first step for our design, we will concentrate on a method that attempts to interfere to only a small degree between the applications and the operating system. We aim to replace the operating system in its role of directly giving network access to applications. Therefore, the *Management Module* of our design will take responsibility for scheduling network access for the applications. This means that it will decide separately for every application about when that the application will start to transmit and receive application data from the network. After the application triggers the middleware the application interacts directly with the operating system. So, by using a simple model we schedule the application's execution, in a way that guarantees service delivery that will satisfy the end user. The output would be the choice of when and which network interface the application will use. This will be done in a way that increases the probability of providing services according to their QoS parameters.

A pseudo code example for these procedures could be the following:

```
/* this is the device data that was previously mentioned and is exploited by the management
module*/
device_data{
        battery;
        CPU;
        memory;
}

get_device_data(){
/* this is the function that collects the device data from the node*/
```

```
}
/* here we will include some of the context data that we will exploit*/
context_data{
        network_link{
                bandwidth;
                cost;
                signal_strength;
        }
        temperature;
        etc....

}
get_contextinfo(){
/* this function is supposed to provide the values that represent the current context*/

return current_context_info;
}
```

The main management function takes as input requests from the application, the context and device data and gives as output policies that will be applied to schedule the applications. Scheduling includes triggering of an application to execute and access a particular network interface.

```
manage(get_contextinfo(), request_subscr(), get_device_data()){

array policies[];
/*this refers to the different policies that apply to each type of application. Every policy refers to a particular combination of the application attributes i, t, d, r, and dl with the context and device data. The policy is associated with the context info, the device data, and the application attributes.*/

app_array[]; //the stack of the applications that are waiting to be given network access

/* periodically, the context info and the device data is checked to see if there are any changes. When there is a change we go through the app_array to see what policies we can apply to the applications based on the new context or device data the creation of policy also takes place every time there is a new subscription for an application. Policies do not refer only to applications that are triggered but also to applications that are waiting. There can be waiting policies. We try to apply policies and schedule the applications in the case of a change in the context or device data and in any case of a new subscription when there is a new subscription a policy is created for the current application. The policy may already exist so it is not essential to be created however when the context parameters change we go though the list of all the subscribed waiting application and if necessary we create new policies for each one*/

/* for the policy creation we need the application attributes, the context and the device data */

create_policy(app x, context_data cd, device_data dd){
/* this is an example of some rules that we can use to create policies */
```

```
/* case of video application that will exploit a WLAN to transmit app data*/
if(cd->network_link->bandwidth == 11Mbps && cd->network_link-
>signal_strength==good)
        if(x->t==1){
                policies[]=(select WLAN) & (triger application);
                apply_policy(policies[x]);
}

if(policy exists){ // if there is already a policy that applies to this case
        apply_policy(policies[i]);
}
else{
        create_policy(policies[]); //create and store a new policy in the policy array
        apply_policy(policies[i]); // apply this policy
}
}
```

It is possible that Middleware might result in worse local resources utilization such as memory and battery. This is because applications may have to reside in the mobile's node memory for a longer period of time until they are opportunistically completed. Moreover, an additional load will be imposed by middleware it self.

3.2.6 Context Module

Context information will be retrieved and stored locally in the mobile node, so that it can be exploited by the *Management Module*. The module that is responsible for this interaction with the context server and the storage of this information is the *Context Module.* It collects and updates the context information that can be beneficial for the applications running on the mobile node and also contributes to this context information by adding information that concerns the mobile node itself. There can also be a *Publishing Module* which sends this information back to the Context Server, that can use it as context information and this information might be requested by other mobile nodes.

3.2.7 Operating System – Virtual Interface

The last part of the middleware's operation is scheduling that applies to all applications. This is done after the application triggers the middleware and it selects the network link that will be used. To implement this all applications use a single interface, this *Virtual Interface* that encapsulates all the different wireless network interfaces. The Middleware can use this virtual interface to provide the enforcement of the wireless link selection and its use. Incoming and outgoing packets go through this interface so that details are hidden from the applications as they only see one interface.

3.3 Design of the Data Pre-Fetching Technique

As we also mentioned above due to the size limitations of this thesis, we provide a more detailed design and analysis **only** for a part of context-aware middleware, which will be the policy of data pre-fetching. Later we also present an evaluation of our suggested design with a demo implementation.

3.3.1 Design Overview

We have designed a mechanism for traffic pre-fetching in an opportunistic way that can be applied to mobile nodes. The data that will be pre-fetched will be anticipated application data that we expect that the mobile device's applications will request in the (near) future. The pre-fetching itself occurs as a low priority system process; therefore it should not compete with regular processes, but it will utilize some of the available resources. The goal is to utilize of inexpensive wireless links when they are available and to avoid negative effects on other users who would also like to have access to the network at that particular time. The opportunistic character of this design refers to the opportunities that can be exploited given current or near term context conditions.

As context information we include the available wireless links at the current time and the number of other network users and the running applications; as well as the device's state concerning its available resources. Figure 3.3 shows the main elements of this design.

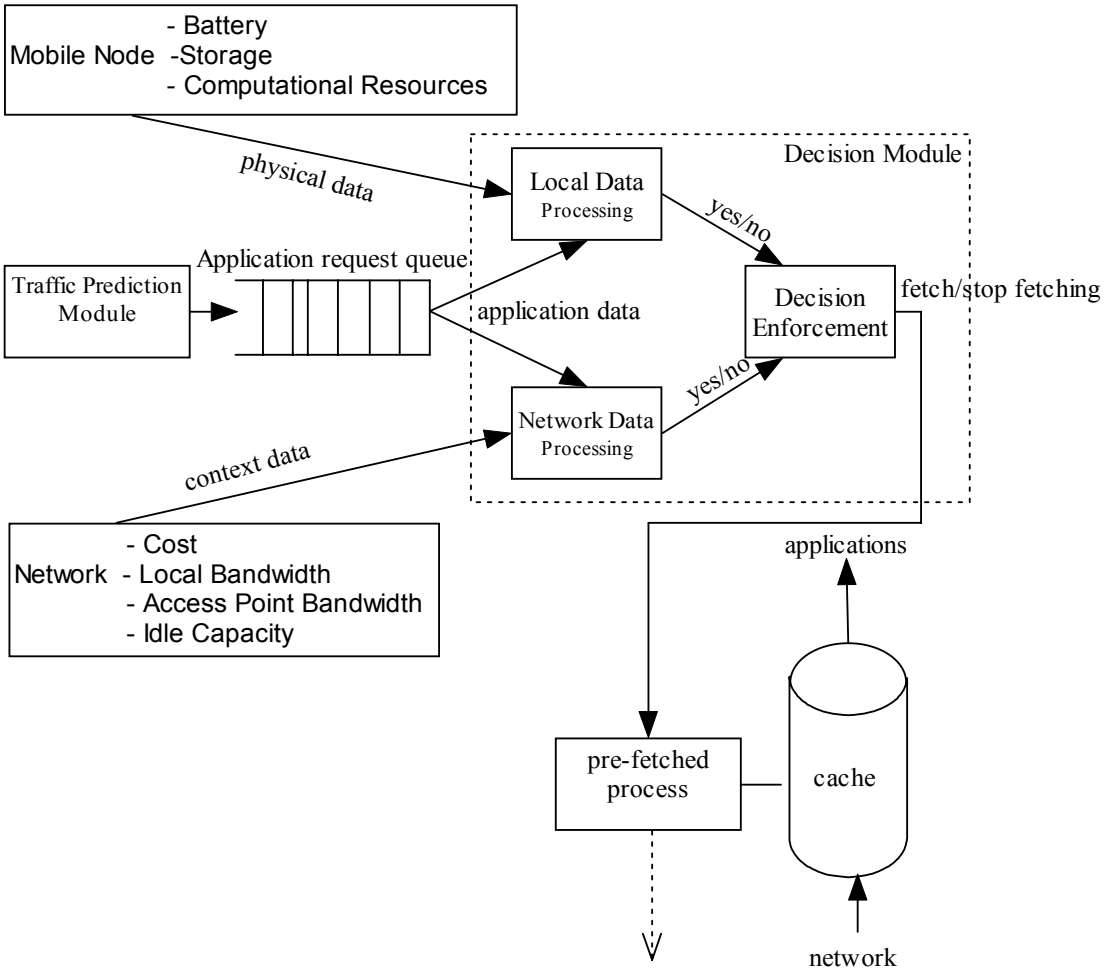

Figure 3.3: Pre-Fetching Design

3.3.2. Mobile Node

The *Mobile Node* entity provides context information regarding some local characteristics of the portable device. These local characteristics are the available battery power hence we can roughly estimate the remaining operational time for the mobile device. Important are also some other information, such as whether the mobile node is connected to an AC adapter or is it running only on the battery. In addition, other significant parameters are the available storage space of the device and the status of other computational resources such as CPU, memory, and hard disk usage. The current status of all these parameters is very important and plays a decisive role in the decision making for pre-fetching. This is because the traffic pre-fetching procedure will consume some of the mobile node's resources and these must be adequate for both pre-fetching **and** for the other the applications that are running. By monitoring the utilization of the mobile device's computational resources we have an indirect view of the status of other competing applications (that run in the same device). Later we will see how these local parameters are taken into consideration and influence the decision making process.

3.3.3. Network Link

Context information that is provided from the network interfaces that are available to the mobile node plays a powerful role for our *Decision Module*. This information mainly refers to attributes of the wireless link such as bandwidth, cost, and idle capacity. The status of the access point, to which this node is connected, is also important in terms of its available bandwidth and idle capacity. This requires consideration also be given to other network users in the same cell. The processing of this context data will later determine whether a particular wireless interface is appropriate or not for data pre-fetching. In the *Decision Module* we will see that a general rule is to select a wireless link with low cost and high bandwidth. However, there is also consideration of its idle capacity as this defines the maximum available bandwidth and can be used to estimate the shortest period for using this link. We can examine this in more details in section 3.3.5 (*Decision Module*).

3.3.4 Traffic Prediction Module

This is the part that is responsible for providing information concerning anticipated requests, which could be preloaded from the network. Data pre-fetching is very closely related to data caching, as it is the same procedure with the difference that in pre-fetching the traffic data should be fetched before it is requested by some application and then it is cached; while usual caching uses the data that have been already retrieved by certain applications and stores it in a local cache memory. Due to this resemblance the prediction mechanism used to cache data can be similar in both cases. Caching can be categorized as client-based or server-based [37]. This separation occurs according to the location where the cached files are stored. In most cases both things happen, as clients which in our case are mobile nodes preserve a cache while web servers (and other proxies) also do so. However, in this thesis we will mainly refer to caching from a client's view.

Future traffic prediction can be accomplished with the use of heuristics that can predict the most probable objects that can be requested by applications. Given this information next we must select those files that are appropriate for caching, e.g., huge files and highly dynamic documents are not advisable to be cached. The method of prediction usually relies on the monitoring of the past traffic as it may suggest the most frequently accessed files if these are suitable for caching. The amount of cached content must **not** result in insufficient storage space for all the files, i.e., there should be a selection among the data that appear to be the most valuable.

An important issue is how to maintain the cache in a way that increases the *cache hit rate* of the files that are requested [37]. There are several algorithms that serve this purpose based upon cache replacement procedures. This is an update technique that attempts to preserve the functionality of the cache; so that it always contains the current most popular files. Some algorithms for cache replacement are LRU, LRU-K [35], FIFO, LRU-MIN, and LFU [38]. Least Recently Used (LRU) and its versions: LRU-K that keeps track of the "k" latest

references to these files and LRU-MIN also considers the size of the document and makes the appropriate comparisons with other cached documents, providing estimations for the documents in the cache that should be replaced by newer ones. First-in First-out (FIFO) and least frequently used (LFU) algorithms also assist in the cache updating. The aim of the cache replacement algorithms is to minimize the cost function with respect to the document size, the time since the file was last accessed, and the transfer time cost [36]. Moreover, there are two main cases of cacheable data: the first are multimedia file such as audio and video files that require lots of storage space and web traffic that contains web documents that have relatively small size and in most cases contain data that remains unchanged for a long time, making it ideal for caching. However, some web pages are marked as **nocache**. This means that the user's web browser will load a "fresh" copy of the page every time he visits it, unlike to the majority of the web pages where they reload after a default expiration time, which is usually few days.

According to other specialized methods application monitoring is involved in order to select the most appropriate to be pre-fetched files. Lei and Duchamp suggest a technique of illustrating/monitoring applications using trees that are called access trees [50]. Every application calls child processes that execute other programs, associated with these programs are using some data files. These trees have references to the files that applications request and they are constructed while the applications are running. A tree is called a *working* tree while the application is still running and *pattern* tree after the application has finished. The files that appear in the *pattern* trees are the files that are considered likely to be requested by future applications and therefore should be pre-fetched.

Another similar method for traffic prediction is "spying" on a user's file access via a utility that the user runs [51]. While this monitoring application is running a list of the accessed files is collected along with the programs that use these specific files so that the sets these files are associated with the corresponding program. A selection of these sets indicates the files that are regarded as predicted to be requested by future executions of these applications.

Development of data prediction techniques is outside the scope of this thesis and therefore we will not offer any solutions or suggestions for this area. There are several existing data prediction algorithms that we apply as input for our middleware design [39], [40].

In our case, the *Traffic Prediction Module* will produce a prioritized list with the specifications of anticipated data (see Figure 3.3).

3.3.5 Decision Module

This is the main module of our design and it is responsible for processing all the context information that is collected and formulating decisions of when and how to utilize the network. Opportunistic communication is driven by the selection of the appropriate time to start applications and when to utilize the network to pre-fetch predicted application data.

3.3.5.1 Network Data Processing

While processing information about the available network links, we first have to estimate the cost of an available link and a determination if it is too expensive to utilize it. The link's available bandwidth together with its idle capacity should be checked next. If the link has a high bandwidth and a significant fraction of its capacity is available then it is an optimal time for our pre-fetching. Additional context information that can be exploited concerns estimating when this wireless link will be available, together with the current and the predicted future location of the mobile node. In this way we can better evaluate whether the present opportunity is a good one or if future conditions will be better to schedule our pre-fetching. It is also essential that we monitor the characteristics of the Access Point of this particular wireless link. This is because we are not only concerned about the resources available to our node but should also consider if other local applications may require bandwidth. Thus we must consider the impact that our network utilization can have on other users that share the network with us. Therefore the decision policy should have as second concern the negative effects that pre-fetching may on other users. Assuming that all the preconditions that we pose are fulfilled, the pre-fetching can start. While this is in progress all the above network conditions should continue to be monitored periodically to detect any changes that will make pre-fetching inappropriate. A decrease in the available bandwidth implies competitive network requests by other applications or users. Considering that the pre-fetching procedure is of low priority, it should not consume network resources that are needed by other applications being executed for this user. When significant other traffic is detected and when the available bandwidth has decreased to a specific low level then the pre-fetching procedure should be halted and postponed until later. When pre-fetching starts again, it can resume from the point that it was interrupted.

Some pseudo-code for the above is:

```
while (estimate_link_cost()<upper_threshold){
        while (detect_idle_capacity()>minimal_reserve_capacity){
        (1)     fetch(predicted_traffic());
        (2)     if(link_busy()){
                        stop_fetching();
                        postpone();
                }
        }
}
//  (1) & (2) are parallel processes
```

3.3.5.2 Local Data Processing

Local resources should also be considered, as they are also utilized by the pre-fetching process and they are shared with other user applications as well. When dealing with mobile devices, battery utilization is one of our greatest concerns; therefore we have to be very

meticulous in its consumption. The prioritized list of the anticipated traffic (Figure 3.3.) provides input to help us estimate what local and network resources are required for this data to be downloaded. The *Traffic Prediction Module* selects elements of data that can be pre-fetched from the network when a suitable opportunity occurs. These requests are sorted in a queue according to its predicted probability of access. Starting from the first element in the queue and taking as extra parameters the current conditions in network and local resources we calculate certain values. Regarding the local resources, we are interested in the size of this anticipated data in the queue so that we can conclude if there is enough available local space for storing it. By considering the network characteristics (i.e. bandwidth) we can also estimate the time that is going to be required for this download to be completed and also the CPU, memory, and disk resources for this download. Assumptions can also be given regarding the amount of battery power that this process will utilize, given that we also can estimate the time that a transfer will take under the current network conditions. By calculating all these different parameters we can decide whether this element of our queue should be downloaded at this time or not. Unless these all preconditions are met, we continue by processing the next element of the queue to determine its suitability.

Pseudo code for this:

```
If(AC_connect()==yes){
        If(Local_Computating_Resources() == available)
                Return OK;
}
Else if(bat_connect()==yes){
        If(battery_capacity() !=low)
                If(Local_Computating_Resources() == adequate)
                        Return OK;
}


/*  all these parameters (“available”, “low”, “adequate”) can be user defined and saved in
user profiles*/
```

4. Evaluation for Opportunistic Traffic Pre-fetching

In this evaluation we will try to predict the performance of our pre-fetching design under various circumstances. We will consider several different scenarios under which pre-fetching can be applied. The aim is to compare the performance of the system with and without the pre-fetching technique.

Predicting the anticipated traffic will **not** be our concern and we will assume that we already have the output of this, i.e., we assume the use of existing traffic prediction algorithms. Thus we will focus on the scheduling of the opportunistic communication.

4.1 Implementation Overview

Real experiments are difficult to realize because of the difficulty in repeating an experiment in exactly the same way. This difficulty limits the accuracy of our conclusions since the experiment's conditions are not exactly the same. As physical tests loose some of their value, we have made some further assumptions. Thus our conclusions are not based on actual measurements and results. The difficulty in repeating the experiment in exactly the same way is due to the fact that it is difficult to have the same network conditions at different time periods and it is also difficult to manually reproduce the same application traffic demands.

Therefore, to simplify our evaluation we utilize a virtual network environment for our experiments - as it can be reproduce the network conditions across multiple runs. This requires the use of tools that emulate or simulate the actual mobile devices and network conditions that we want to have in our tests.

In our case we are investigating the behavior of mobile nodes in wireless networks, so as a first step we need to emulate the mobile node’s is a virtual network that should include one or more wireless network interfaces. Later we will simulate some applications that these mobile nodes will run. For this we will use another tool (flowreplay [54], tcpreplay [42]) that will replay previously captured application traffic.

To create reproducible network conditions we will use the NIST NET network emulator [34]. In order to do this we have to first decide upon the characteristics of the wireless interfaces we will have so that we can enter the appropriate parameters into NIST NET.

4.2 Capturing Network Traffic

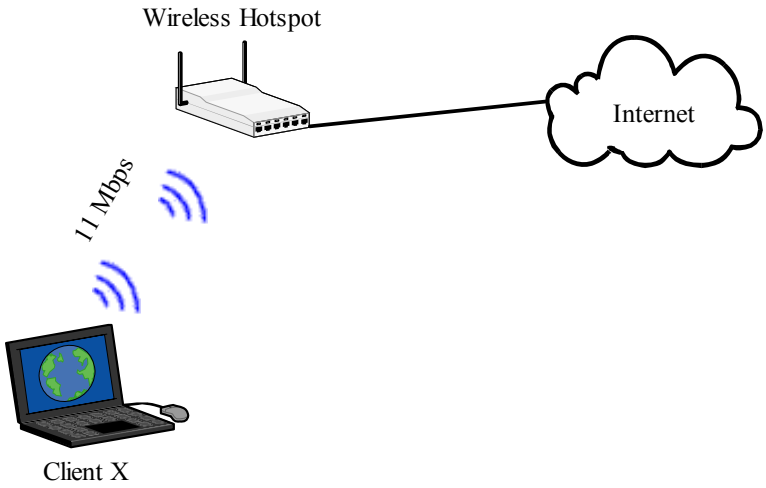

Figure 4.1: Traffic Capture Topology

As we also mentioned above, in order to perform realistic experiments we will use traces captured from real network applications. Therefore we must first capture some network traffic that we later "replay" for our experiments. For this we used a network sniffer tool called Ethereal [41]. The node in which we will run this tool for capturing the network traffic is "Client X" as shown in Figure 4.1. It is a notebook computer that was equipped an IEEE 802.11b WLAN interface operating at a maximum speed of 11 Mbps. At "Client X" we run several network applications such as instant messaging with "MSN Messenger", a Voice over IP application "Skype"[45], real-time video streaming with "Real Player", web surfing with the web browser "Firefox", e-mail sending through an SMTP server using e-mail client application ("Thunderbird"), as well as an ftp client application for photo uploading, and downloading of new virus definitions. While these applications are running we monitor the network with the "Ethereal" tool for approximately 18 minutes (1080 seconds). After this, we have the traffic captured as a trace file (*.pcap).

4.2.2 Captured Application Time Schedule

By analyzing the trace file (*.pcap) of the captured traffic we compute some statistics as shown in Figure 4.2. This figure depicts how the application data are distributed over the time we monitored this user's traffic and it also shows the duration of the communication on a peer application basis. This information is available through *Ethereal* filters, which allows us to filter each application's traffic and separate it from the total traffic. This trace file includes time stamps for each packet, thus we can extract time stamps for each application's packets.

	start	end	times
Web Surfing	3 sec	– 1035 sec	
Instant Messenger (MSN)	9 sec	– 1074 sec	
Voice over IP application Skype	112 sec	– 803 sec	
Watch video (online TV)	244 sec	– 700 sec	
Ftp upload	328 sec	– 488 sec	
Download Virus Definitions	670 sec	– 817 sec	
First Email through SMTP	441 sec	– 547 sec	
Second Email through SMTP	713 sec	– 856 sec	

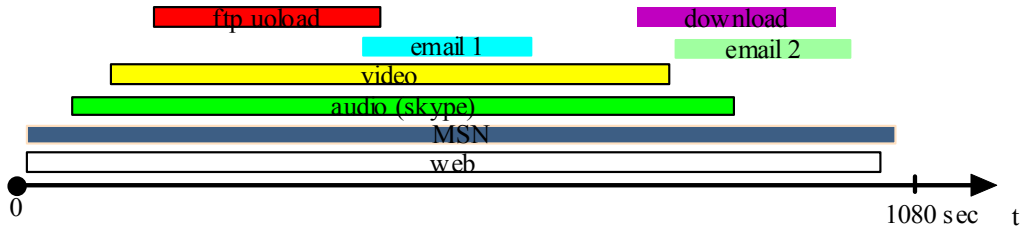

Figure 4.2: Captured Traffic Time Schedule

As it can be seen from the figure the mobile node is running several applications in parallel. The user of "Client X" is surfing the web, while at the same time he is chatting with a friend of his thought "MSN Messenger". After a while he receives a call via "Skype" from a member of his family that currently lives abroad. He accepts this call and starts the Voice over IP application. A bit later during this conversation, his relative recommends that he watch an online TV channel with an interesting documentary about antiques. Therefore the user starts the online video streaming application. Later, a connection is established with a remote ftp server where the user uploads his photos when the local memory storage reaches its maximum levels. Following this is a download from a remote server of the new virus definitions for the antivirus application that is running in the mobile node. In the mean time he also sends two emails with attached song files to a friend of his, through an SMTP mail server.

In the above case study the traffic that can be anticipated is primarily the upload and download of files. The upload of the photos is a regular procedure for the user, as the user wishes to upload photos to an ftp server, whenever the memory of the user's digital camera is full or approaching full. Similarly, the update of the antivirus definition is also a download that is performed by many users that use the same antivirus program and traffic could be stored in local proxies. Because these applications are not interactive or delay sensitive, this traffic streams could be considered for pre-fetching.

4.2.3 Bandwidth Charts of Captured Traffic

A further analysis of the captured traffic gives us information about the bandwidth distribution of the application data over time. This can be seen if the following *Figures 4.3*, *4.4* and *4.5*.

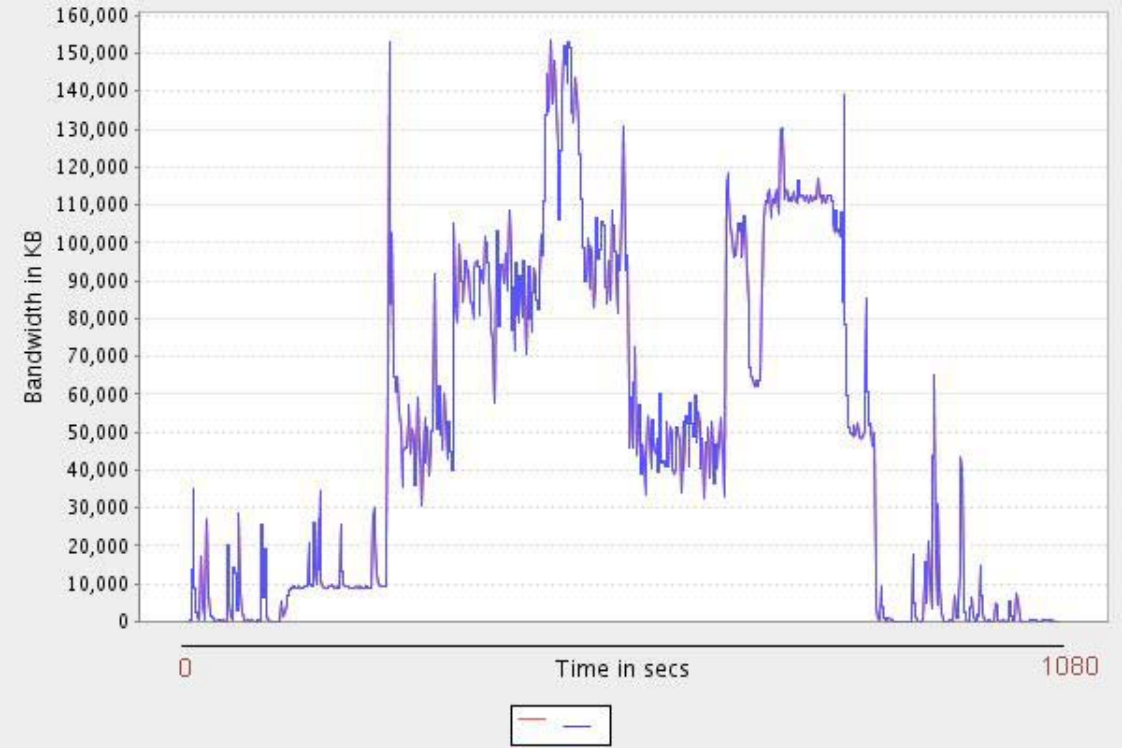

Figure 4.3: Total Captured Traffic – Bandwidth chart

Figure 4.3 shows the bandwidth chart of the total captured traffic.

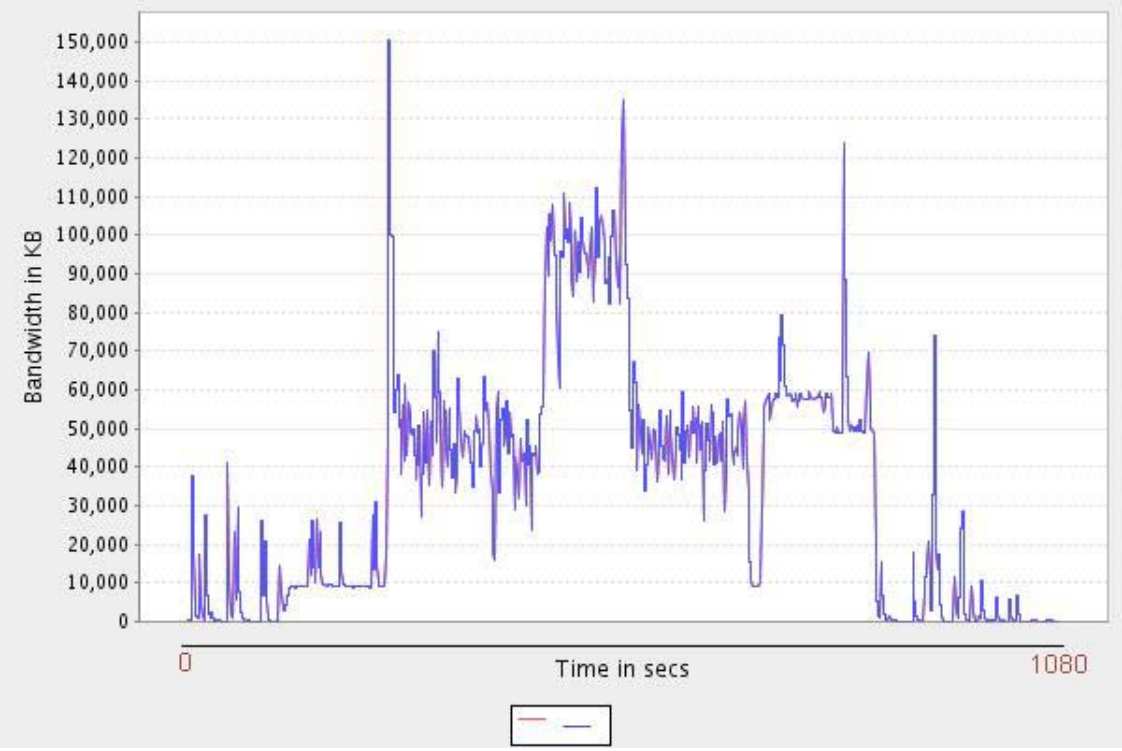

Figure 4.4: Total Captured traffic excluding potential "pre-fetched" Traffic – Bandwidth chart

Figure 4.4 shows the total traffic after we subtracted the ftp upload and download streams that we regards as traffic that can be pre-fetched.

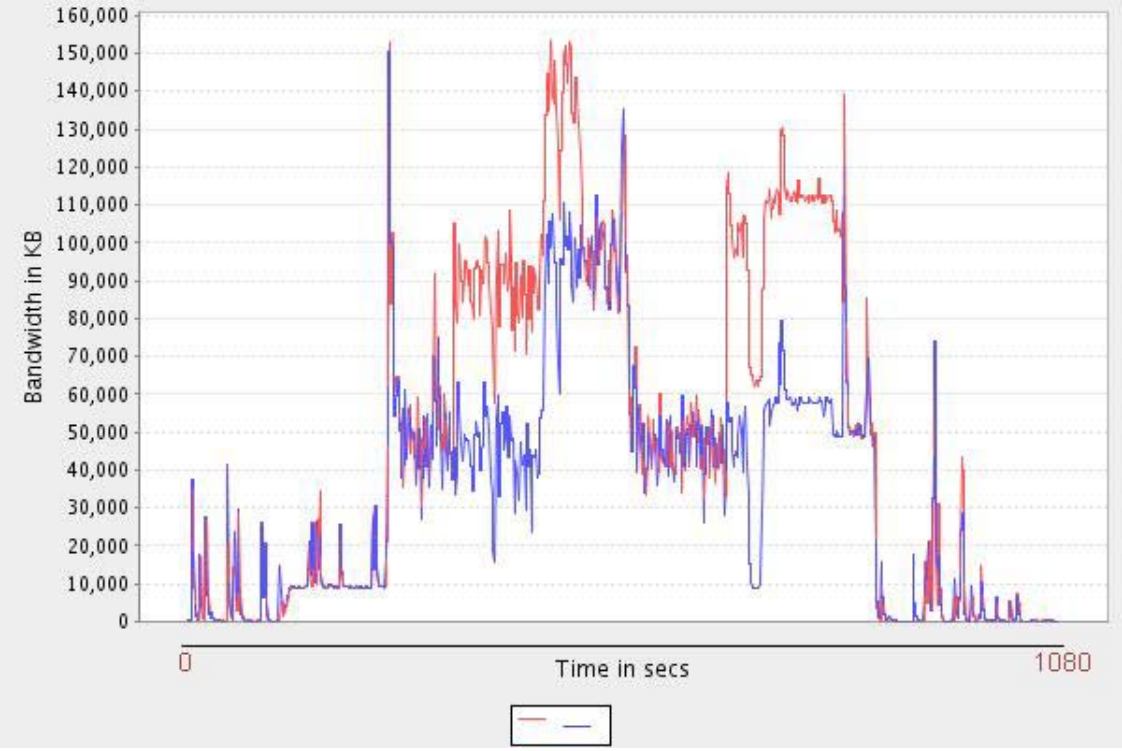

Figure 4.5: Traffic chart - combing (Figures 4.3 and 4.4)

Figure 4.5 shows in one chart both sets of traffic (i.e., as were previously shown in figures 4.3 and 4.4). By examining this chart we can see that the bandwidth distribution is not very flat, so this means that traffic could have been scheduled in a different way to achieve better overall optimization of the network utilization. This issue will be addressed by scheduling anticipated traffic in an efficient way.

4.3 Creating Network Conditions

The next step after capturing the application network data will be to replay it with different parameters. However, before reproducing the captured network traffic we need to create the network conditions that we use for our scenarios. In Figure 4.6 we depict the network topology that we will emulate.

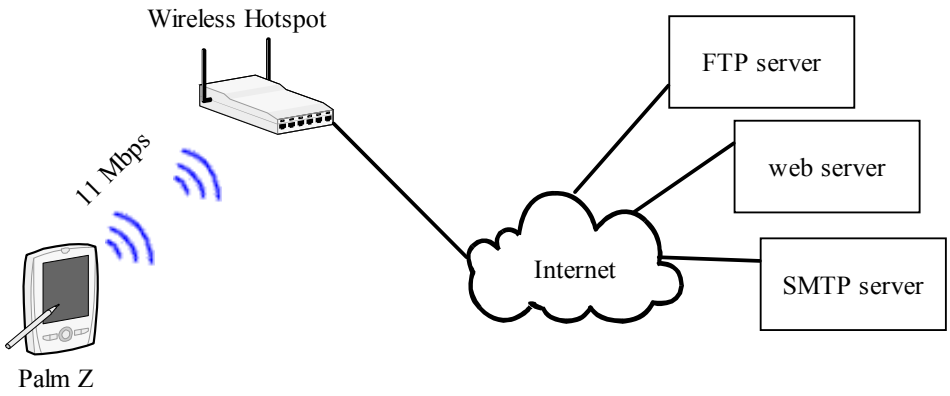

Figure 4.6: Network Topology

In order to emulate the above network we will use the NIST NET network emulator tool [34]. NIST NET emulates the desired network after configuring the tool with the appropriate parameters. Figure 4.7 shows the actual network topology using the NIST NET emulator; here "Client X" node represents a "Palm Z" node. An example of the NIST NET parameters that can describe a particular network condition are the following:

source node:	192.168.8.70	
destination node:	192.168.8.71	
delay:	65 ms	
delay delsigma:	5 ms	
bandwidth:	1441792	(11 Mbps)
drop percentage:	1 %	
duplication percentage:	0.3 %	

The fields of "source" and "destination" node specify the link for which the following settings will be applied. The "delay" parameter refers to the one-way delay time between these two nodes (65 ms). "delay delsigma" refers to the variation of the "delay" parameter. The "bandwidth" parameter specifies the bandwidth of the link between the two nodes in bytes per second; in our case this equals 1441792 (bytes per second). "drop" and "duplication" percentage refers to the packets that can be dropped or duplicated. The decision upon the values of the parameters was done by measuring these characteristics with *ping* [43] Linux command in a cell of the WLAN of Stockholm University with average load of users (approximately 25).

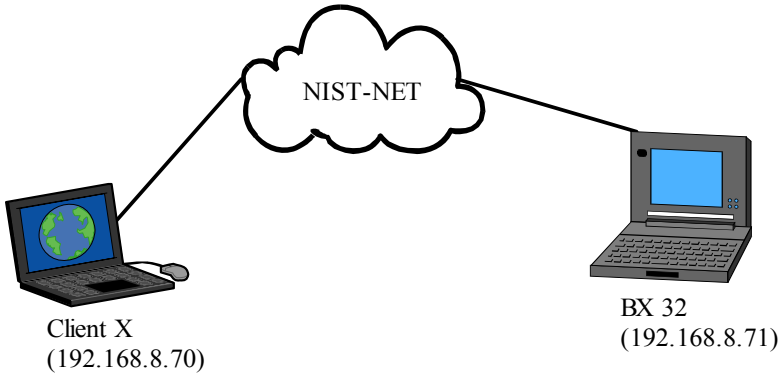

Figure 4.7: NIST-NET Emulation

4.4 Simulating Network Servers

We are almost ready to start replaying the captured traffic back to the network. But before this we should set up certain servers that will listen and respond to the client's part of the replayed traffic. For this reason we simulate two FTP and two SMTP servers for interacting with the FTP upload, download and the sending of two e-mails respectively. The server side is represented by a modified version of the tool *netcat* [55]. In our scenario, the servers are established by listening to a TCP port for the client application to connect; and they are also set up to have an average delay response of some milliseconds, which simulates a server's behaviour due to the load of multiple users.

4.5 Reproducing Network Traffic

Now that the network conditions are created, we are ready to start to transmit traffic. We begin with the traffic trace file that our network monitor tool Ethereal provided to us. This captured traffic can be replayed in the network with the use of an appropriate tool. For this we used modified version of *flowreplay* [54] and *tcpreplay* [42]. These are tools that generate network traffic according to trace files of Ethereal (*pcap* files). Our captured traffic describes several kinds of applications. Some of them are delay and interactive sensitive (Real-time video, Voice Application) and require a fixed transmission rate reserved for them, while others such as ftp upload or download will compete for resources (i.e., bandwidth) as they are not real-time/interactive applications.
However, first the *pcap* file has to be edited in order to be able to be retransmitted in the emulated network. Therefore the headers of the captured packets have to be edited so that they have as endpoints the two nodes of our network ("Client X" and "BX 32"). This was done using the *tcprewrite* tool [44].

As we also mentioned above in this scenario we will use as the anticipated traffic the upload and download traffic. Taking this into account we will apply our pre-fetching policy to the total traffic. Then we will see how our policies can affect the overall performance of the network link.

4.6 Policy Enforcement

By applying some of the scheduling principles described in section 3.3.5 to the traffic we can compare the overall traffic shaping with and without policy enforcement; so as to evaluate the advantage of exploiting the network link more intelligently. The parameterization of the values in all the following policies is simply indicative for our particular experiments. In a practical implementation, these parameters could come from a user specified configuration, be specified by the system administrator or the network provider.

4.6.1 Idle Link Traffic Policy

According to this policy we monitor the link's utilized bandwidth, looking for opportunities when the link is rather idle, to trigger the transmission of anticipated data. The network link is viewed in our policy as rather idle when the bandwidth utilization is less than 6 KB/s for at least a period of 6 seconds.
When such low utilization is detected, based on monitoring the link every 2 seconds, i.e., if all three values satisfy the condition of being lower than 6 KB, this moment is regarded as an **opportunity** for triggering the data pre-fetching procedure. The goal of this policy is to trigger the fetching of the anticipated traffic before it is requested by its application. Then, the pre-fetched application data can be requested from the **cache** that pre-fetching preserves. There is also the case when there was no opportunity for some anticipated data to be pre-fetched but the data is now required by the application. There are two possible ways to address this case. The first is to give immediate access to the application to start utilizing the network. Whereas, the second approach is to continue applying the *Idle Link Traffic Policy,* trying to find for an opportunity to perform **post-updating** at this time.

4.6.2 Local Resource Driven Policy

As we also mentioned in sections *3.3.2* and *3.3.5.2*, one of the preconditions for pre-fetching is the availability of sufficient local resources of the mobile node such as battery, Cache, CPU, Memory, and possible Disk usage. In this case a good **opportunity** for pre-fetching occurs when the mobile node is AC connected or its battery capacity is over 20% and the size of the Cache is more than the size of the data to be pre-fetched and the CPU usage is under 85% and the utilized Memory is less than 85% and the Disk Usage is less than 85%. This can be easier understood by the following logical statement.

(AC || battery>20%) && (Cache>sizeof(pre_data)) && (CPU<85%) && (Memory <85%) && (Disk <85%)

The above is both a precondition for triggering pre-fetching and it should also remain true while pre-fetching is performed; otherwise the pre-fetching should be stopped and postponed.

4.6.3 Network Driven Policy

Similarly for the Network, there is a precondition that should be observed when scheduling opportunistic traffic. This is described by the following logical expression:

(cost<10 price units) && (available_bandwidth>51200 bytes) && (available_time>timefor(pre)) && (access_point_load<90 Mbps)

Where "cost" it is the pricing that the network provider would charge this user for the excepted traffic. This cost could be time based, traffic based, or even flat rate. "available_bandwidth" is the maximum available bandwidth that the network provider will give to this mobile user. "available_time" is the estimated time that this link will be available to the user; it has to be sufficient for the anticipated data to be up/downloaded. To calculate the time required for an up/download we take the available to the maximum bandwidth and the time the download starts, together with the size of the data to be downloaded and we calculate the time that is needed. Usually the actual time needed is greater that the calculated time, because while the pre-fetching may start when the link is idle - later there can be traffic competing for this link. By "access_point_load" we refer to the total traffic load that the access point experiences. In this parameterization it should be less than 90 Mbps so that we avoid negative impact to other users that use the same access point.

4.6.4 Competitive Applications Driven Policy

Once pre-fetching is triggered there is still a need to continue monitoring the link for competing traffic, in order to ensure that the pre-fetching appropriate or if it needs to be stopped. This occurs when other user applications compete for the link's bandwidth. In our implementation we require that at least 5 KB/s of bandwidth **remain** of the total maximum bandwidth, if we are to perform pre-fetching. When this is false, which means there are other applications that require high bandwidth, pre-fetching should be stopped and postponed.

4.7 Test Scenario

In this scenario we assume that a user (Eric) is going from his office to a restaurant in a nearby shopping center to have dinner with a friend. For his transportation he uses a public train. While he is in this train, in order to pass the time he turns on his mobile device and he runs the applications as they were described in section *4.2*. This is a simulated scenario where we utilize the previously captured application traffic. So, there are two types of anticipated traffic; these are, the ftp upload and download as they were mentioned in section *4.2.2*.

During Eric's trip to the shopping center there are available several different wireless hotspots between which the mobile node has to choose and makes hand-offs, in order to provide continuous connectivity to these applications. Here we assume that the mobile node utilizes four different wireless hot spots, whose characteristics and duration are described in *Table 4*. In order to emulate these network conditions we use the NIST NET tool that is

described in section *4.2.4*. The parameters that are used to emulate the four different wireless hot spots are shown in *Table 5*.

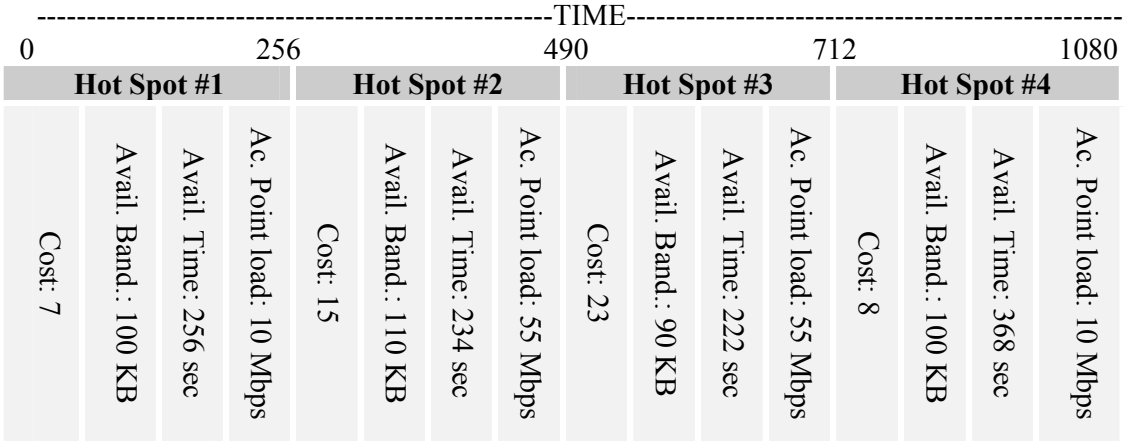

Table 4: Case 1 - Hot Spots

	Hot Spot #1	**Hot Spot #2**	**Hot Spot #3**	**Hot Spot #4**
Delay/delsigma	65/10 ms	75/10 ms	70/20 ms	50/15 ms
Drop	1%	0.5 %	0.7 %	0.8 %
Duplication	0.3 %	0.1 %	0.5 %	0.4 %
Bandwidth	100 Kb/s	110 Kb/s	90 Kb/s	100 Kb/s

Table 5: Hot Spots #1, #2, #3, & #4

Our design does not dictate choices of hotspots or the hand-offs between different wireless networks, rather this selection of the hotspots as well as the hand off to the next hot spot is done by other mechanisms. Our implementation simply utilizes the current wireless hot spot to which the node is connected. The transition from one network to another is assumed to be instantaneous and occurs at the times indicated in *Table 4*.

Eric uses his device for 18 minutes (or 1080 seconds). *Table 6* shows the status of the device's local resources for each sub-period.

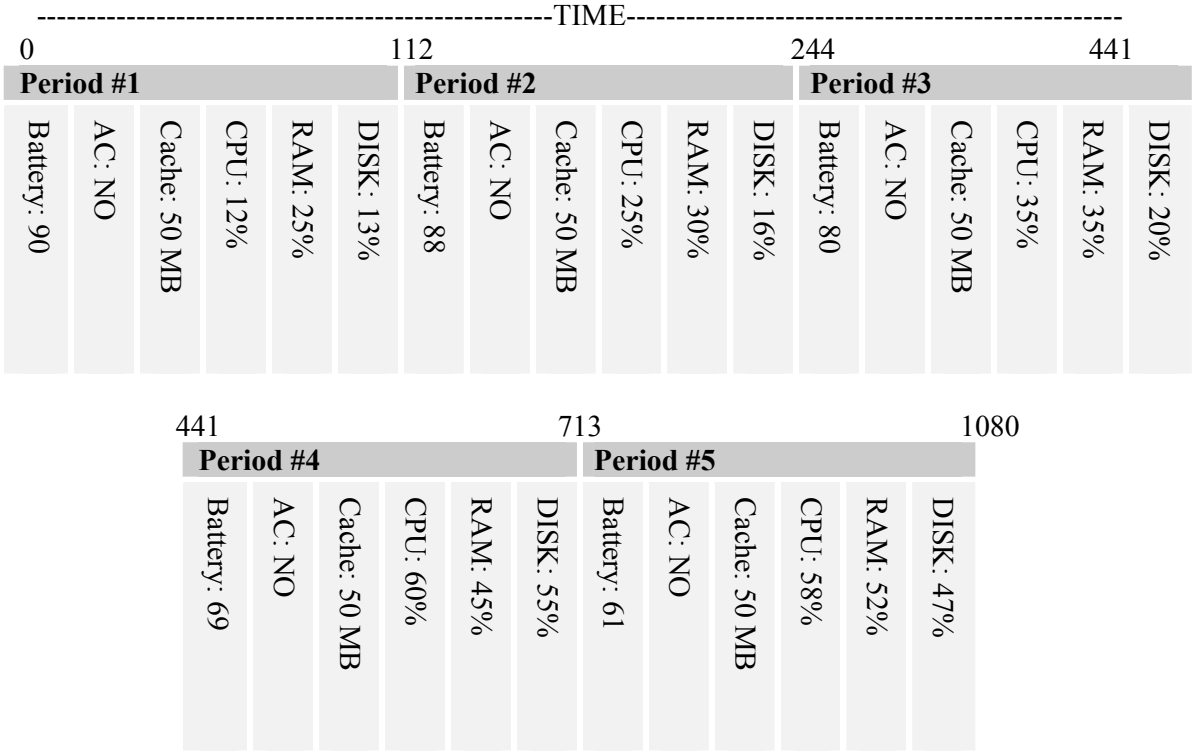

Table 6: Local Resources' Values

Table 7 shows the average response times of the servers. Some statistics that result from measurements can be seen at *Table 8*. They present the association between the response time for the server and the bandwidth that it gives for the client.

Servers	FTP 1	FTP 2	SMTP 1	SMTP 2
Delay (ms)	110	100	85	75

Table 7: Case 1: Servers' Delays

Delay in ms	Bandwidth in KB/s
120	~ 52
100	~ 60
80	~ 70
70	~ 85
60	~ 95

Table 8: Server Delay/Bandwidth relation

First we examine each application's traffic without applying any of our policies. *Figure 4.8* shows the bandwidth utilization of the produced traffic as a function of time. The application's execution schedule was described in section *4.2.2*. In *Figure 4.8* shows the time when the applications are executed and their duration. Considering "ftp download" (pre-fetched stream #2) there is a discontinuity in its transmission which is thought to be due to the server's lack of response for that period.

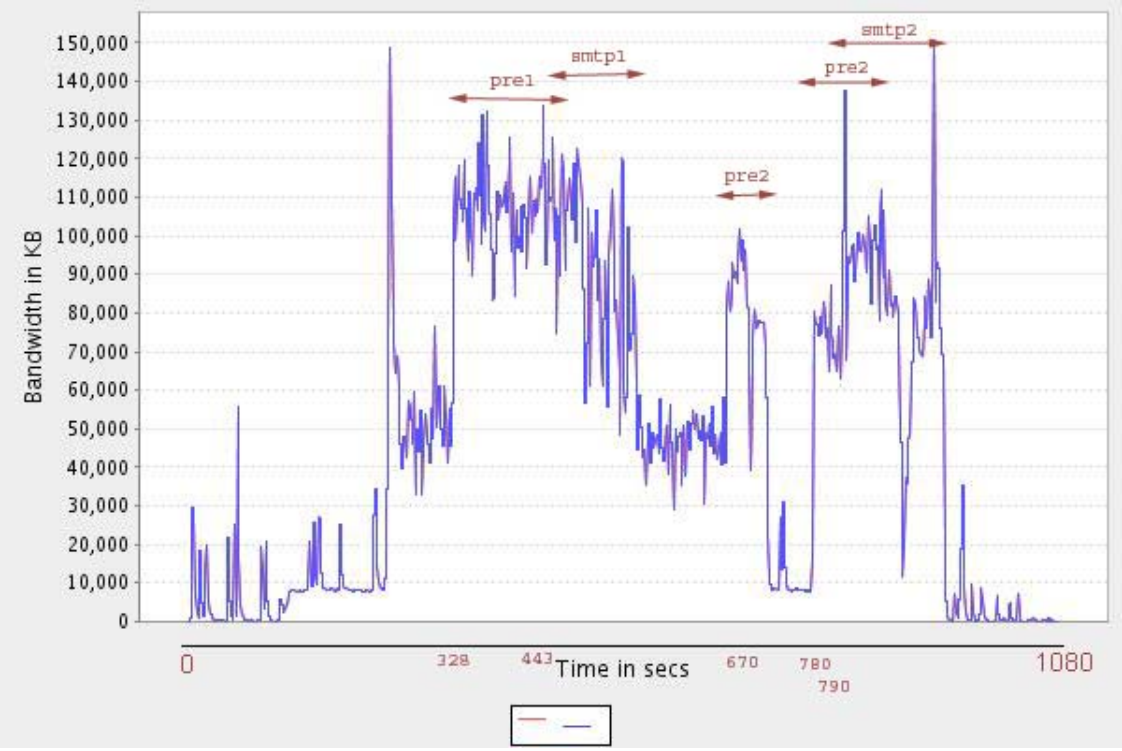

Figure 4.8: Case 1: Traffic – Bandwidth Chart

Now we repeat the experiment in exactly the same way, but this time applying our policies as they were described in sections *4.6.1 – 4.6.4*. The resulting bandwidth chart is shown in *Figure 4.9*. As we can see the anticipated upload traffic is triggered to start utilizing the network at the 14th second when the first good **opportunity** occurs. Our *Idle Link Policy* together with optimal network and local resource conditions detects this **opportunity** and triggers the pre-fetching of the first traffic stream to start. This traffic connects with the first FTP server and continues until a point where the server stops responding for a while, but then it continues at around the 192nd second and the pre-fetching finishes successfully. The FTP server with its 110 milliseconds average response delay provides us with an average bandwidth of 58 KB/s. Another opportunity is requested for the pre-fetching of the second stream of the anticipated data. This opportunity occurs at around the 838th second when the link is again rather idle and the pre-fetching starts and finishes successfully after a while. In this second case the anticipated traffic transmission takes place later than requested by the application. Therefore it is more properly called *post*-updating rather than pre-fetching. Furthermore, since it is not a time critical data transfer, our policy does not inconvenience the application or lessen the user's experience.

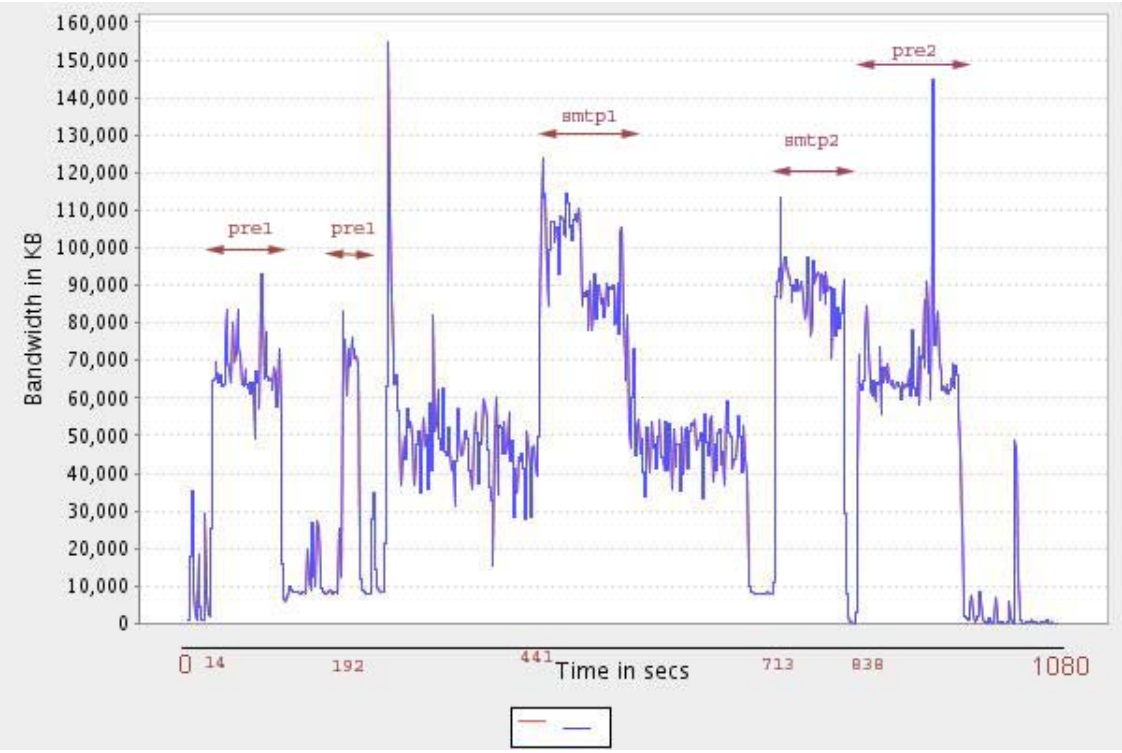

Figure 4.9: Case 1: Traffic (with pre-fetching) – Bandwidth Chart

Figure 4.10 is a combination of *Figures 4.8* and *4.9*. Here we can clearly see and compare the differences in traffic shape with and without our pre-fetching policies. Our original traffic (shown in blue) that was reproduced as it was captured is contrasted with the same traffic after we pass it through our policy enforcement (shown in red). As we can see the overall traffic load is lower when we apply our policy and the average high bandwidth utilized is lower than before. The result is more efficient network utilization.

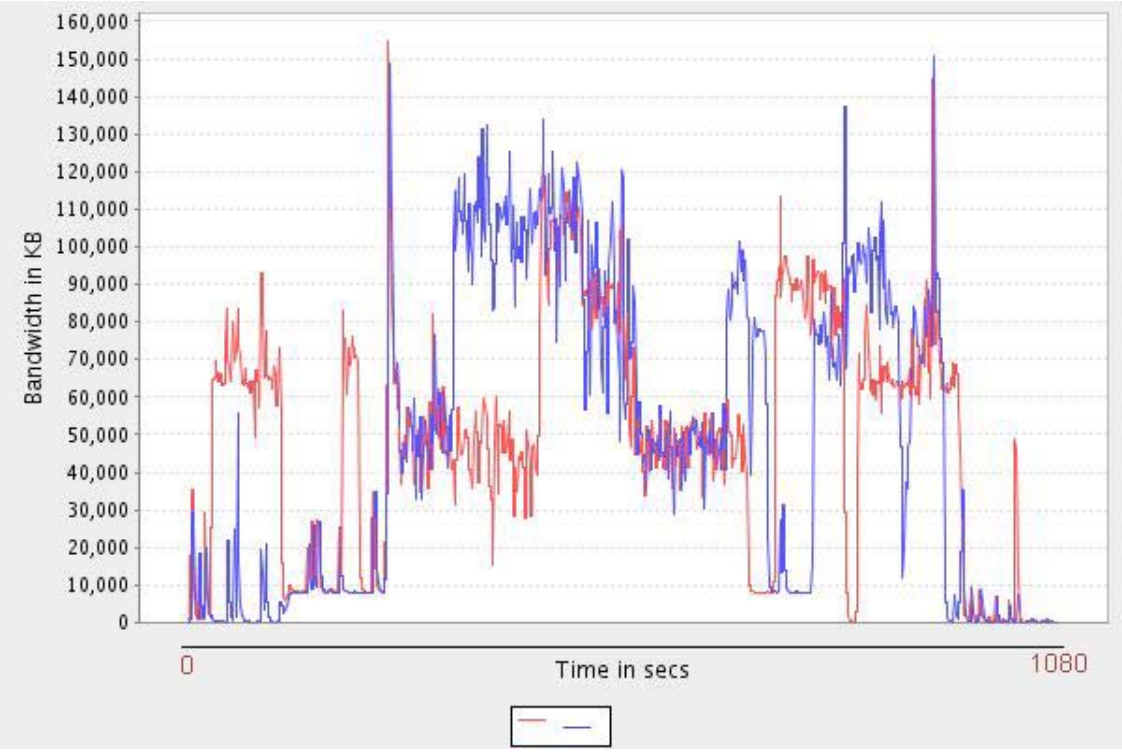

Figure 4.10: Case 1: Dual Traffic Chart (Figures 4.8 and 4.9)

Next we consider a second experiment where we alter only a few parameters. These are the parameters that are highlighted in *Tables 9* and *10*. Note that these changes decrease the delay associated with FTP server_1 and increase the available bandwidth.

Servers	FTP 1	FTP 2	SMTP 1	SMTP 2
Delay (ms)	60	100	85	75

Table 9: Case 2: Servers' Delays

----------TIME----------															
0			256				490				712				1080
Hot Spot #1				**Hot Spot #2**				**Hot Spot #3**				**Hot Spot #4**			
Cost: 7	Avail. Band.: 115 KB	Avail. Time: 256 sec	Ac. Point load: 10 Mbps	Cost: 15	Avail. Band.: 110 KB	Avail. Time: 234 sec	Ac. Point load: 55 Mbps	Cost: 23	Avail. Band.: 90 KB	Avail. Time: 222 sec	Ac. Point load: 55 Mbps	Cost: 8	Avail. Band.: 100 KB	Avail. Time: 368 sec	Ac. Point load: 10 Mbps

Table 10: Case 2 - Hot Spots

We repeat the experiment as before and we examine the equivalent results. *Figure 4.11* is the equivalent *Figure to 4.8* and it also very similar to it.

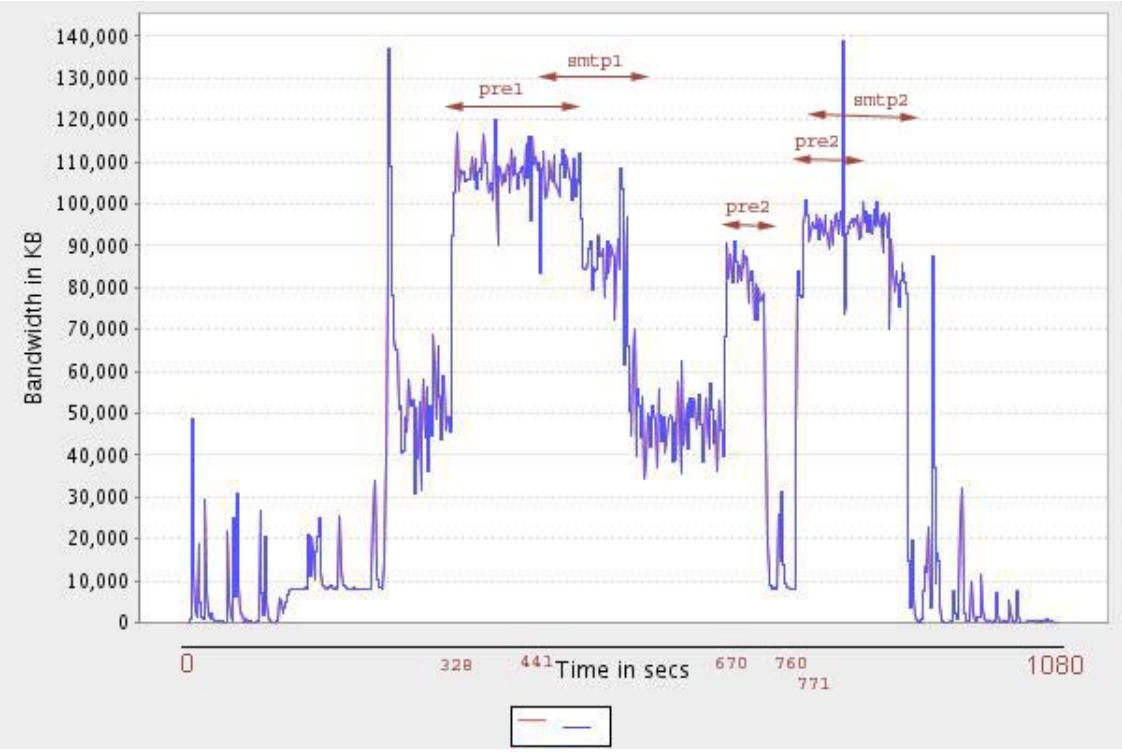

Figure 4.11: Case 2: Traffic – Bandwidth Chart

Whereas, *Figure 4.12* is equivalent to *Figure 4.9*. An **opportunity** for triggering the first pre-fetched stream is found at the 15th second. Because in this case our FTP server has a faster respond time (60 ms) and the first host spot gives us a higher (115 KB/s) available maximum bandwidth, the first pre-fetching stream finishes its transmission earlier than in previous time. Shortly after this, while the link is still rather idle a second opportunity is found for the 2nd pre-fetching to start transmitting. This happens at the 114th second, but after a while it is stopped due to the server's inactivity and it continues again at the 146th second. All this time

the conditions, that enabled an opportunity for pre-fetching, continue to be monitored. At the 249th seconds we violate certain conditions required for pre-fetching at this moment. This is determined by the *Competitive Applications Driven Policy* that was described in section *4.6.4*. This is because the application requiring "online video" is executed and it required considerable bandwidth that was not available if we were to continue pre-fetching. Therefore, pre-fetching is suspended and the Cache size reserved for it is increased based upon the size of the anticipated data that we were about to downloaded. A new opportunity continues to be sought until it is found at the 847th second. The present version of our implementation does not support resuming an interrupting pre-fetched traffic, so the 2nd pre-fetched stream has to be downloaded from the beginning.

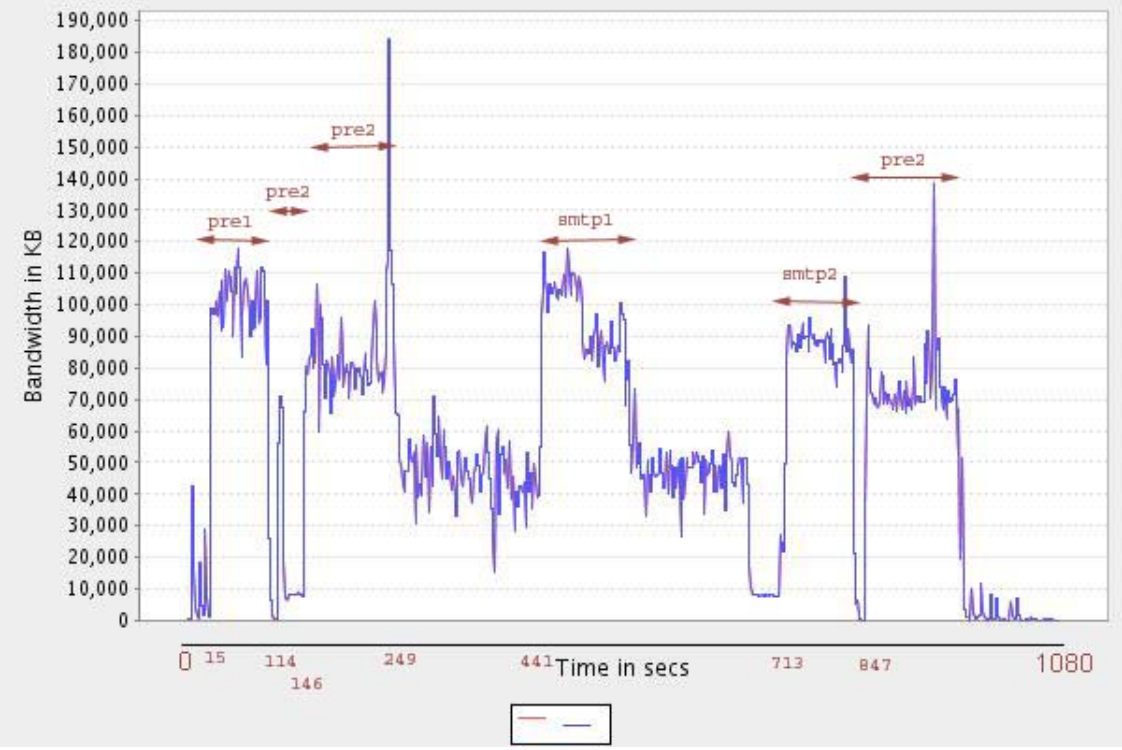

Figure 4.12: Case 2: Traffic (with pre-fetching) – Bandwidth Chart

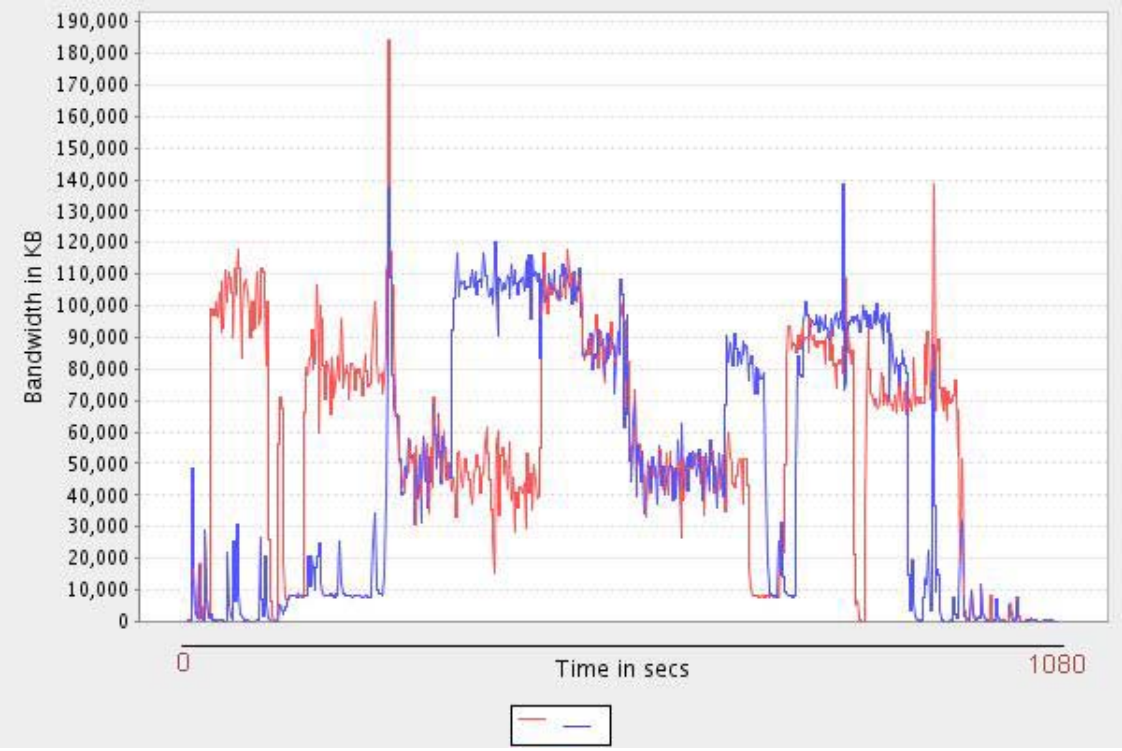

Figure 4.13: Case 2: Dual Traffic Chart (Figures 4.8 and 4.9)

Figure 4.13 combines both *Figures 4.11* and *4.12* in a single chart. With blue color is shown the traffic without policing and with red with our policies applied. We can conclude that there is a better traffic shaping with the pre-fetching policies. However, in this case, the failure in pre-fetching the 2nd anticipated transfer increases the total traffic. This increase seems not to influence the other applications running on the mobile node. But it is regarded as unnecessary and increases the general traffic load of the network provider, which can have a negative impact on other network users or even to the user himself (if he is charged based upon the amount of traffic he generates).

4.9 Implementation Limitations

The simulation of the local utilized resources such as Battery, CPU, and Memory is not the best possible. We don't have the actual applications running and therefore we assume the utilization of local resources that they would have in our system. The supplementary tools that we use to produce the applications' transmitted traffic simply simulate the network behavior of the applications and give the values for the network parameters. However, their utilization of local resources does not reflect the real applications. For instance, *flowreplay* and *tcpreplay* while running use maximum speed of CPU (100%) and they do not demand much of the memory's capacity.

The current version of the DEMO implementation takes into consideration in its *Decision Module* **only** the current context conditions. There are also near term (future) context conditions that should be considered. This means that there should be a decision for every moment that is regarded an **opportunity** to determine if it is the best behaviour for the next

period. *Network* and *Local Data Processing* should predict context values as well. If the output of this process gives a prediction for a *better* opportunity in the near future, then pre-fetching should not be triggered but rather should wait to examine the future context information at a later time.

5. CONCLUSIONS AND FUTURE WORK

5.1 Summary

In this thesis we proposed a Middleware design that exploits context information and supports opportunistic communication for applications running in mobile nodes. This Middleware was designed to be transparent from the user and to assist in continuous-ubiquitous service provisioning by providing the expected QoS for each application; it is also designed in a way that does not interfere with protocols such as TCP.

Middleware policy enforcement focuses on scheduling the application traffic and selecting the appropriate network interface for transmitting this data. The expected result is the efficient utilization of the network resources that are available to the mobile user. The main concern is the efficient management of the network resources, which in some cases is done at the expense of inefficient utilization of the mobile node's local resources such as memory, and storage capacity, that generally are regarded as ample and increasing in capacity.

Initially, a generic Middleware architecture was proposed for dealing with the general case of opportunistic behavior. The context information regarded as most significant for the mobile node is the available networks and their characteristics. Applications requesting network access subscribe to the Middleware through its *Subscription Module*. The device's local resources are monitored, together with context data in order to enable the *Management Module* to create scheduling policies for the applications that will be stored in the *Policy Repository* and also applied to the applications themselves. These policies provide control of application scheduling as well as selection of the appropriate network interface for each application.

Later, a more detailed analysis was contacted for the opportunistic communication case of *Data Pre-Fetching*. *Data Pre-fetching* resembles the general case of Middleware design in the way context information and local sources status are managed by the *Decision Module*. An additional component for the pre-fetching design was the *Traffic Prediction Module* that predicts the data that needs to be pre-fetched. A *Cache* is also used to store the pre-fetched data. Several policies are applied by the *Decision Module*. Via the *Idle Link Traffic Policy* the network's link utilized bandwidth is monitored for idle conditions that will indicate an opportunity for pre-fetching to be triggered. The *Local Resource Driven Policy* examines whether the mobile node's local resources are sufficient to allow potential pre-fetching. The *Network Driven Policy* evaluates network conditions to find a time for an opportunistic communication. While *Competitive Applications Driven Policy* monitors the network link during pre-fetching in order to allow competing applications to access the network and stops the pre-fetching procedure if necessary.

5.2 Conclusion

A question might occur at this point is: Is middleware useful for a mobile system? A correct-certain answer can only be given after testing such middleware in the real world and evaluating its performance. The possible results from the enforcement of pre-fetching part of Middleware are the following:

a) Cases where pre-fetching is not applied. Applications are not influenced.
b) Cases where it is applied and produces **better** results for the applications than in case (a).
c) Cases where it is applied and produces **worse** results for the applications than in case (a).

Middleware can be regarded as beneficial when (b) cases are more frequent than (c) and (a) cases. Even in (a) the middleware is regarded as inefficient, since we load our mobile system by running an extra software component (Middleware) that does not take any useful action.

However, our evaluation indicates that the pre-fetching design presents some expected positive results. The design seems to work satisfactory and we can conclude that middleware in general is beneficial for a mobile system, as it provides more efficient network utilization. Nevertheless, our testing scenarios were picked in a way that illustrates well Middleware's functionality and worthiness, but resemble only a small part of the real cases that a mobile device might encounter.

Middleware's performance also depends to a large extent on its configuration. For instance, pre-fetching's configuration refers to the values that we use as thresholds for the local resources, network data, and idle capacity. These values indicate when an opportunity for pre-fetching will occur. An optimal assigning of these values should result in good opportunistic behavior for Middleware.

5.3 Future work

The next step is the implementation of this Middleware's design in a mobile system and its connection with other entities such as a Context Server. Middleware's context interface should be developed to communicate with and collect context data from the Context Server. Middleware's *Management Module* should then be able to extract and process the context data that it receives from the Context Server in a predefined format [28].

Context-Aware Middleware's current development supports management of context information that is used by context-aware services. Middleware could also assist the distribution of context information that is accumulated from the mobile node itself. For example, this can refer to the values describe the Device's Sate that are collected by Middleware and could be useful for others. A way of exchanging of context information is proposed by Context Data Exchange Protocol (CDXP) [26]. Collection and formulation of the context information is usually the task of the Context Server, but in this case mobile node can act as a sensor.

Development is also required by the part of the Middleware that accumulates the state of the Local Resources (Physical Data). This refers to resources such as Battery Capacity, CPU, Disk Capacity, …, that should be determined from the system via appropriate local sensors.

There should also be a more detailed design and evaluation regarding the case where there are multiple network interfaces available for an application and therefore there should be an appropriate selection among them. In our design we mention a Virtual Interface that connects the operating system with the available network interfaces. For this, there should be a decision making mechanism with criteria similar to the general case of opportunistic communication.

As we mentioned before, what facilitates a good opportunistic behaviour for Middleware is its configuration. This can be done by each user individually and stored in his profile or it can be in the jurisdiction of the system administrator. However, what will enable the best parameterization for the configuration could be an automatic re-configuration by the system itself. An artificially intelligent system can be used to configure Middleware. This system could utilize a learning mechanism that compares results from past configurations and proposes future configurations.

Moreover, a more detailed definition of the way that *Data Buffering* works should be proposed. This is the technique used for providing applications with execution continuity when there are insufficient network resources. Then application's data is buffered locally so that it can be sent later when sufficient network resources are available again.

The *Traffic Prediction Module* of the pre-fetching technique also requires further development. For pre-fetching to be effective it is very important that traffic predictions are as accurate as possible. Otherwise, Middleware might waste network resources.

Some other Middleware policy that needs to be investigated is a policy that violates the QoS requirements of the applications, but provides connectivity to all of the applications. This can be achieved by shaping the application traffic that will give reduced quality for every application, but it will not create competition for network resources between the applications.

Middleware is currently designed to manage certain type of context information. However, there is context information that is excluded from this set and might be useful for several applications running in mobile nodes. This information could include variables such as temperature, light level, and location. A future Middleware design could also include this additional context information.

REFERENCES

[1] Giulio Mola. *"Interactions of Vertical Handoffs with 802.11b wireless LANs: Handoff Policy"*, Master of Science Thesis at KTH Microelectronics and Information Technology, Stockholm, Sweden March 2004.

[2] Maria R. Ebling, Lily B. Mummert and David C. Steere. *"Overcoming the Network Bottleneck in Mobile Computing"*, School of Computer Science Carnegie Mellon University, Santa Cruz, CA, December 1994.

[3] M. Satyanarayana*." Mobile Computing",* School of Computer Science Carnegie Mellon University*,* Santa Cruz, CA, September 1993, Vol. 26, No. 9.

[4] Xin Liu. *"Opportunistic Scheduling in Wireless Communication Networks"*, A Thesis Submitted to the Faculty of Purdue University, West Lafayette, Indiana, US, December 2002.

[5] Carl-Gustaf Jansson, Martin Jonsson, Theo Kanter, Fredrik Kilander, Gerald Maguire, Li Wei and Andreas Wennlund. *"Context Data Distribution Concepts and Approaches in the ACAS Project"*, Wireless @ KTH, Stockholm, Sweden May 2004.

[6] Alois Ferscha University of Linz *ferscha@soft.uni-linz.ac.at*, Wolfgang Beer University of Linz beer@ssw.uni-linz.ac.at and Wolfgang Narzt University of Linz narzt@soft.uni-linz.ac.at. *"Location Awareness in Community Wireless LANs"*. Paper at the Informatik 2001, Vienna, Austria, September 2001.

[7] Andreas Wennlund. *"Practical Context-Aware Adaptive Communication: Using Service Classification and Service Decomposition"*. Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Stockholm, Sweden, May 2004

[8] Samsung Official web site: http://www.samsung.com/PressCenter/PressRelease/PressRelease.asp?seq=20040907_0000069353, accessed January 2006.

[9] Srikant Sharma Inho, Baek Yuvrajsinh Dodia and Tzi-cker Chiueh. OmniCon: *"A Mobile IP-based Vertical Handoff System for Wireless LAN and GPRS Links"*. Computer Science Department, Stony Brook University
Stony Brook, NY, August 2004

[10] Gerald Q. Maguire Jr.. Personal communication regarding future mobile devices. Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH). September 2004

[11] Francois Willame*. "Opportunistic multihop communication using mobile platforms for very sparse infrastructures"*. Thesis Proposal, Radio Communication Systems, KTH, Stockholm, Sweden, November 2003.

[12] Stavros A. Xynogalas, Maria K. Chantzara, Irene C. Sygkouna, Stavros P. Vrontis, Ioanna G. Roussaki, and Miltiades E. Anagnostou, *"Context Management for the Provision of Adaptive Services to Roaming Users"*, IEEE Wireless Communications, April 2004, pp. 40-47

[13] Andreas Wennlund. "*Requirements for a Support System for Context-Aware Adaptive Communication in Heterogeneous (Wireless) Networks"*. Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Stockholm, Sweden, May 2004

[14] W. Schilit. *"A System Architecture for Context-Aware Mobile Computing"*. Ph.D. thesis, Columbia University, US, 1995.

[15] Cellular Network Optimisation based on Mobile Location, http://www.telecom.ntua.gr/cello/. Accessed October 2004

[16] D. Garlan *et al.*, *"Project Aura: Towards Distraction-Free Pervasive Computing",* IEEE Pervasive Comp., vol. 1, no. 2, 2002, pp. 22–31.

[17] K. M. Hansen *et al.*. *"Hypermedia in the Kimura System: Using Spatial, Temporal, and Navigational Relationships to Support Multitasking and Background Awareness"*. ACM Hypertext Conf. 2001, Aarhus, Denmark, August. 2001, pp. 14–18.

[18] G. Chen and D. Kotz, *"Solar: A Pervasive-Computing Infrastructure for Context-Aware Mobile Applications,"* Tech. rep. TR2002-421, Department of Computer Science, Dartmouth College, February 2002.

[19] M. Ebling, G. Hunt, and H. Lei, *"Issues for Context Services for Pervasive Computing,"* Wksp. Middleware for Mobile Comp., Heidelberg, Germany, November 2001.

[20] K. Henricksen, J. Indulska, and A. Rakotonirainy, *"Modelling Context Information in Pervasive Computing Systems,"* 1st Int'l. Conf. Pervasive Comp., Zurich, Switzerland, Aug. 2002, *LNCS*, vol. 2414, pp. 167–80.

[21] CONTEXT, Active Creation, Delivery and Management of Efficient Context Aware Services. IST-2001-38142-CONTEXT. http://context.upc.es/. Accessed October 2004

[22] Symbian OS Official web site, http://www.symbian.com/technology/technology.html. Accessed October 2004

[23] Konstantinos Avgeropoulos. *"Service Policy Management for User User-Centric Services in Heterogeneous Mobile Networks"*. Master of Science Thesis at Microelectronics and Information Technology Department, KTH, Stockholm, Sweden, March 2004.

[24] George Y. Liu. *"Efficient Mobility Management for Wireless Mobile Computing and Communications"*. Licentiate Thesis at Telecommunication System Laboratory, Department of Teleinformatics, Royal Institute of Technology, Stockholm, Sweden, March 1995

[25] C. Komar, and C. Ersoy, *"Location Tracking and Location Based Service Using IEEE 802.11 WLAN Infrastructure"*, European Wireless 2004, Barcelona Spain, 24-27 February 2004

[26] Andreas Wennlund. *"Context-aware Wearable Device for Reconfigurable Application Networks"*, Master of Science Thesis at KTH Microelectronics and Information Technology, Stockholm, Sweden April 2004.

[27] Kerry Jean, Kun Yang, and Alex Galis. *"A Policy Based Context-aware Service for Next Generation Networks"*. Department of Electronic & Electrical Engineering, University College London, Torrington Place, London WC1E 7JE, UK. September 2003

[28] Asim Jarrar. *"Context Server support for opportunistic and adaptive mobile communication"*. Master of Science Thesis at KTH Microelectronics and Information Technology, Stockholm, Sweden June 2003.

[29] Mark T. Smith and Gerald Q. Maguire Jr., SmartBadge/BadgePad version 4, HP Labs and Royal Institute of Technology (KTH), http://www.it.kth.se/~maguire/badge4.html, date of access 2004-11-02.

[30] Roberto Gioacchino Cascella. *"Reconfigurable Application Networks through Peer Discovery and Handovers"*. Master of Science Thesis at KTH Microelectronics and Information Technology, Stockholm, Sweden June 2003.

[31] M. Satyanarayana*." Mobile Information Access",* IEEE Personal Communications, Vol. 3, No. 1, February 1996

[32] Roque, Soares, and Oliveira, *"VESPER Project - Validation of VHE Concept"*, University of Porto, Porto, Portugal, 2001

[33] Khalaily Mahdee and Hijazi Tarek, *"NIST Lan Simulator"*, Dept. of Electrical Engineering, Israel Institute of Technology, Israel, Spring 2003

[34] NIST NET network emulator Official web site, http://www-x.antd.nist.gov/nistnet. Accessed June 2005

[35] Elizabeth J. O'Neil, Patrick E. O'Neil, Gerhard Weikum. *"The LRU-K Page Replacement Algorithm For Database Disk Buffering"*. Department of Mathematics and Computer Science University of Massachusetts, Boston, US June 1993

[36] Liangzhong Yin, Guohong Cao, Ying Cai. *"A Generalized Target-Driven Cache Replacement Policy for Mobile Environments"*. Department of Computer Science and Engineering, University of Pennsylvania, USA, March 2003

[37] Igor Tatarinov, Alex Rousskov, Valery Soloviev. *"Static Caching in Web Servers"*. Computer Science Department, North Dakota State University, USA, September 1997

[38] Stephen Williams, Marc Abrams, Charles Stanbridge, Ghaleb Abdulla, Edward Fox. *"Removal Policies in Network Caches for World Wide Web Documents"*. ACM Sigcomm96, Stanford, USA, August 1996

[39] Fan Yang, Zhaozheng Yin, Henry X. Liu, Bin Ran. *"An On-line Recursive Short-term Traffic Prediction Algorithm"*. Department of Civil and Environmental Engineering, University of Wisconsin at Madison, USA, June 2002

[40] Hao Chen, Ljiljana Trajkovic. *"Trunked Radio Systems Traffic Prediction Based on User Clusters"*. School of Engineering Science, Simon Fraser University Burnaby, Canada, September 2004.

[41] Ethereal: A Network Protocol Analyser, Official web site http://www.ethereal.com/, Accessed June 2005.

[42] Tcpreplay: Pcap editing and replay tools for *NIX, web site http://tcpreplay.sourceforge.net/, Accessed June 2005.

[43] Ping: The ping Manual Page, web site http://www.stopspam.org/usenet/mmf/man/ping.html, Accessed September 2005.

[44] Tcprewrite: Manual, web site http://tcpreplay.sourceforge.net/man/tcprewrite.html, Accessed September 2005.

[45] skype™: Voice Over IP application, web site http://www.skype.com/helloagain.html, Accessed October 2005.

[46] Ivar Gaitan. *"Real-time services and multi-hop networks"*, Master of Science Thesis at KTH Microelectronics and Information Technology, Stockholm, Sweden September 2005.

[47] George Y. Liu. *"The Effectiveness of a Full-Mobility Architecture for Wireless Mobile Computing and Personal Communications"*. PhD Thesis at Computer Communication Systems Laboratory, Telecommunication System Laboratory, Department of Teleinformatics, Royal Institute of Technology, Stockholm, Sweden, March 1996.

[48] George Liu. *"Description of MMP Algorithms"*, Ericsson Report, (T/B 94:229), May 1994.

[49] Alvin Yew, Christos Bohoris, Antonio Liotta and George Pavlou. *"Quality of Service Management for the Virtual Home Environment"*, Centre for Communication Systems Research,School of Electronics, Computing & Mathematics, University of Surrey, Guildford, Surrey, GU2 7XH, UK, 2001

[50] Hui Lei and Dan Duchamp. *"An Analytical Approach to File Pre-fetching"*, Computer Science Department, Columbia University, New York, January 1997

[51] Carl Tait, Hui Lei. *"Intelligent File Hoarding for Mobile Computers"*, Mobile Computing and Networking book, 1995

[52] Thomas M. Kroeger and Darrell D. E. Long. *"The Case for Efficient File Access Pattern Modeling"*, Jack Baskin School of Engineering, University of California, Santa Cruz, January 1996.

[53] Johannes Jansson. *"Context-aware Service Allocation in Personal Area Networks"*, Master of Science Thesis at KTH Microelectronics and Information Technology, Stockholm, Sweden November 2004.

[54] Flowreaply: Flowreplay Design Notes, web site https://www.synfin.net/papers/flowreplay.pdf, accessed November 2005.

[55] Netcat: The GNU Netcat – Official homepage, web site http://netcat.sourceforge.net/, accessed December 2005.

COS/CCS 2006-1

www.kth.se