UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

Design and Implementation of a Digital Video Recorder, with Live Video Streaming to Cellphone, over Mobile Broadband

Johann Stegmann

Thesis presented in partial fulfilment of the requirements for the degree
Master of Science in Electronic Engineering
at the University of Stellenbosch

Supervisor: Dr. R.Wolhuter
March 2007

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: ....
J.P. Stegmann

Date: ....

Opsomming

Die inhoud van hierdie Tesis handel oor die toenemende vermoë van die mobiele Internet die moontlike gebruik van selfone om video monitering stelsels aan te vul. Die klem van hierdie Tesis val op die intydse lewering van video beelde na selfone.
Die eienskappe, vermoëns en beperkings van die netwerke en selfone word ondersoek, asook die moontlike metodes om intydse video beelde na selfone te lewer. Die ontwerp en implementering van ’n digitale video monitering stelsel word aangebied. Twee weergawes van die video lewering is ontwerp en gebruik in twee sagteware weergawes vir selfone.
Die evaluasie en toets van die stelsel word aangebied en moontlike toekomstige navorsing en verbeterings word voorgestel.

Abstract

The work presented in this Thesis relates to the increased capabilities of the mobile Internet and the possible use of cellphones as an enhancement to video surveillance systems. The focus of the Thesis is on the delivery of live video content to Java enabled cellphones.

The various characteristics, capabilities and limitations of the mobile networks- and phones are investigated. Various options for streaming video content to cellphones are also explored. The design and implementation of a digital surveillance system with the ability to stream live video to a cellphone is presented. Two versions of the streaming protocol are developed and implemented in cellphone applications, with which the video stream can be viewed.

An evaluation and real-life testing of the applications are presented. Recommendations regarding further enhancements are provided.

Acknowledgements

Many thanks to my supervisor, Dr. Riaan Wolhuter, for his encouragement and expert advice.

More thanks; to Tania, Wayne, all my friends and family who have supported and encouraged me, especially in the last couple of months.

Very special thanks to my parents for their encouragement and everlasting dedication and love over the years.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMINOLOGY

B_{eff}	Effective throughput of the communications channel
t_x	Frame transmission time
t_i	Inter-frame interval
3GPP	3rd Generation Partner Project
API	Application programming interfaces
CDC	Connected Device Configuration
CIF	Common Intermediate Format
CLDC	Connected Limited Device Configuration
DNS	Domain Name Service
DVR	Digital video recorder
EDGE	Enhanced Data Rates for GSM Evolution
fps	Frames per second (framerate)
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication
GUI	Graphical User Interface
H.263	Low-bitrate video codec
HSDPA	High Speed Download Packet Access
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IGMP	Internet Group Management Protocol
IP	Internet Protocol
ISO	International Standards Organization
ITU	International Telecommunications Union
J2EE	Java Enterprise Edition
J2ME	Java Micro Edition
J2SE	Java Standard Edition
JAM	Java Application Manager
JAR	Java Archive File
JPEG	Joint Photographic Experts Group
JVM	Java Virtual Machine
kbps	kilobits per second
MDI	Multiple Document Interface
MIDlet	Main class of an MIDP application
MIDP	Mobile Information Device Profile
MJPEG	Motion JPEG
MMAPI	Mobile Media API
MPEG	Moving Picture Experts Group
ms	milliseconds
OSI	Open Systems Interconnection
PC	Personal computer
PNG	Portable Network Graphics
QoS	Quality of Service
RAM	Random Access Memory
ROM	Read-Only Memory
RTCP	Real -time Control protocol
RTP	Real-Time Transfer Protocol
RTSP	Real-time Streaming protocol
SMS	Short Message Service
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

UMTS	Universal Mobile Telecommunications System
VCL	Visual Component Library
VoIP	Voice over IP

1 Introduction and Thesis Objectives

The recent surge in the amount of internet- and Java enabled cellphones on the market has opened up numerous possibilities, in which cellphones can be used, to enhance the quality of life for the average individual. These devices can access basically any type of content on the internet. The mobile Internet has evolved from providing meager text-based web pages, to being a resource, rich with multimedia content.

As the capabilities of cellphones improved, so did the mobile connection capabilities of both the phones and the mobile carriers, up to a point where cellphone networks can be used for video and audio streaming, instant messaging services, voice over IP, games and more. The improvements in the processing power and capabilities of cellphones also improved to such an extent that almost limitless possibilities for new applications are available.

The high crime rates and increased tendency of people to want to feel secure and to protect their belongings and interests have brought an increase in the amount of video surveillance systems used. A very large number of businesses- and an ever increasing number of private properties are nowadays protected by video surveillance systems. These systems can be greatly enhanced, if it can be monitored from anywhere, any time the business- or home owner wishes to. This is an area where the cellphone has particular potential.

Currently, the ability to stream video to cellphones is mostly limited to newer handsets and the video content is mostly provided by content providers who want to make a profit, from delivering the content. Private- and small scale video streaming to cellphones is however still in its infancy and very few applications exist which can stream video from an ordinary PC to a large variety of Java enabled cellphones.

This thesis presents the considerations-, design- and implementation of a digital video recorder (DVR), which can stream live video images to Java enabled cellphones.

1.1 Thesis Objectives

This thesis presents an exploration of the mobile Internet environment, as well as the application development environment of cellular phones. The various characteristics, capabilities and limitations of the mobile networks- and devices are investigated. The various options for streaming video content to cellphones are also explored. The information is ultimately used in the design- and implementation of a digital video recorder (DVR), which can stream live video images to Java enabled cellphones.

The main objectives of this thesis are therefore:

- To research the characteristics, limitations and capabilities of the mobile networks- and devices.
- To research the options available for implementing video streaming to cellphones
- To use the knowledge in the design of a digital video recorder with the ability to stream live video to as great a range of cellphones as possible. The DVR application must be able to run on an ordinary Windows PC, with cameras connected to the PC. The DVR must provide a user friendly interface and allow a certain measure of security against access from unauthorized cellphones. The DVR must be able to do video motion detection and to send a notification when motion is detected.

These objectives were successfully met and demonstrated.

1.2 Thesis Overview

Chapter 2 provides information and research, regarding the capabilities and limitations of modern java-enabled cellphones and mobile networks. The basic principles of digital networks, network models, data transport protocols, video streaming methods and application development environments are presented. The relevance of each item to the thesis objective is highlighted.

Chapter 3 presents the theoretical design of the DVR, cellphone application and streaming video protocol. A communications model is first derived, followed by the theoretical design of the video streaming protocol. The high-level flow diagrams of the DVR- and cellphone applications are then presented.

Chapter 4 presents the implementation of the designs, derived in Chapter 3. Two versions of the cellphone application are implemented. The issues encountered in the implementation stage are noted and the workarounds used discussed.

Chapter 5 presents the evaluation of the project, against the Thesis goals. Screenshots of the DVR and cellphone application are provided, along with the results of testing the mobile application on different handsets.

Chapter 6 concludes the Thesis by giving a summary of the contributions made by the thesis, results and presenting suggestions for future work.

2 Background and Research

The first step in designing the mobile streaming surveillance solution is to understand and compare the various video streaming algorithms and related network protocols. This chapter describes the basic principles of digital networks, network models, data transport protocols, video streaming methods and application development environments. Since various permutations of video streaming methods, data networks and transport protocols exist, the most common streaming techniques are described and compared, with relevance to the structure and abilities of a typical cellphone network- and user device. The relevance of the various underlying protocols and techniques to the design of the streaming video application is also discussed.

The chapter kicks off by describing the models for a digital communications network, with insight into how the different building blocks of the networks fit together. An overview is then given of each of the building blocks, starting at the lowest level moving up to the highest level, which is the end-user application.

2.1 Network Models

Any communication between two different devices need to use some kind of communications medium, as well as a method (protocol) to exchange information between the devices. In order to define the communication between different devices more clearly, a network model is used. This section gives an overview of the two most common network models used in modern communications systems, and the relevance to GSM type networks.
The two common models, on which most modern networks are based, are the ISO OSI Model and the Internet Protocol Suite. These networks models and the relevance of the implementations thereof, to this project, are discussed in the following paragraphs.

2.1.1 Open Systems Interconnection (OSI) Reference Model

The OSI reference model was developed in 1984, by a working group under the International Standards Organization (ISO). The reference model was developed to provide a standard reference framework, for the way in which different information devices communicate.

The OSI model defines different stages, in a data communications network, through which data must pass through to get from one device to another. The stages are defined by seven OSI Communication Layers. [1] Each layer is dependant on the functions of the layer below it and only provides functions to the layer above it. In other words, each layer receives data from the layer below it and prepares the data for use by the layer above it.

Each layer provides a specific function in the transmission of data between two devices in the link. Each layer in turn communicates with the same layer on the other end of the network connection, with a common protocol for each layer. The OSI model provides a set of guidelines to implement the protocols of each layer, to ensure compatibility between different networked devices, provided that the devices adhere to the OSI guidelines.

Figure 2.1-1 illustrates the interaction of the layers of the OSI model.

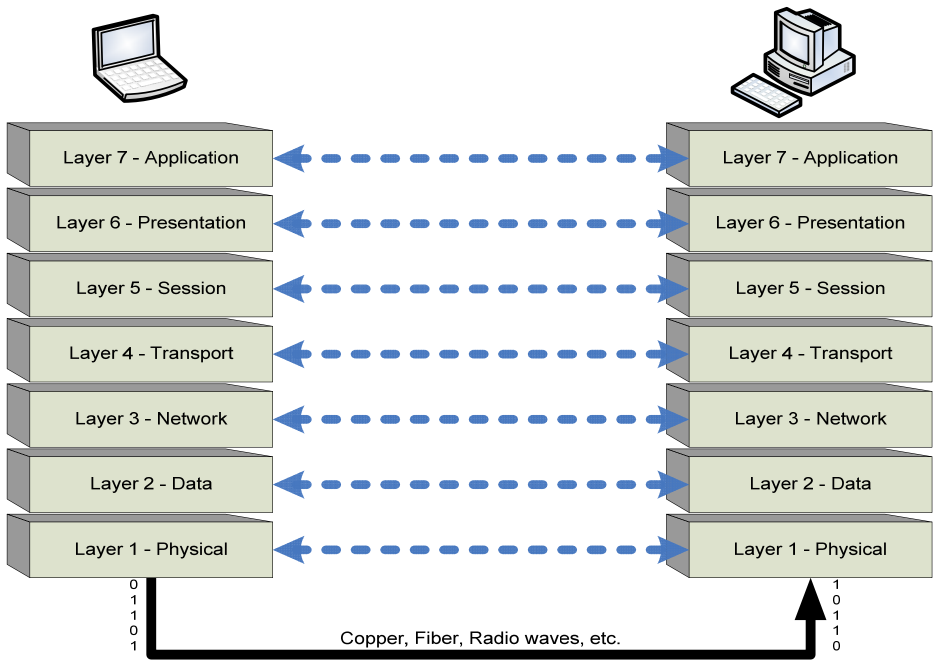


Figure 2.1-1: Layers of the OSI Network Model

A more detailed description of each of the layers in the illustration can be found in the ISO standard document, ISO/IEC 7498-1:1994

Although the standard aims to ensure interoperability between all networked devices, from different vendors, the full functionality of the standard was never fully implemented in the Internet, although various subsets of the model are utilized. [2] The standard that better defines the way the Internet works is more commonly known as the Internet Protocol Suite, or Internet Reference Model.

2.1.2 Internet Protocol Suite (Internet Reference Model)

The Internet protocols were first developed by the Defense Advanced Research Projects Agency, for communication between dissimilar computer systems, at research facilities. Initially, the Internet Reference Model was defined to have 4 layers, but this model has evolved to a five-layer model, according to more recent documents and text books. No formal publication or documentation exists which specifically define the model as a

Standard, however. The more recent, 5-layer model, is discussed in the subsequent paragraphs.

The Internet Protocol Suite has less rigidly defined layers than the OSI Model and is thus more suited to real-world applications but, although the layers of the Internet model have somewhat different definitions from the OSI model, it spans the complete range of OSI layers. It is therefore in a way similar but also dissimilar to the OSI model.

With the inclusion of TCP/IP and also more common applications, for example Email and file-transfer in the Internet Suite definitions, the Internet Protocols have become the foundation on which the World Wide Web (Internet) is based. [3] The Internet Protocols are widely used in communication systems and is implemented by the majority of modern networked communication devices.

Figure 2.1-2 illustrates the layers and relevant protocols of the Internet Model, as well as how it typically compares with the OSI model.

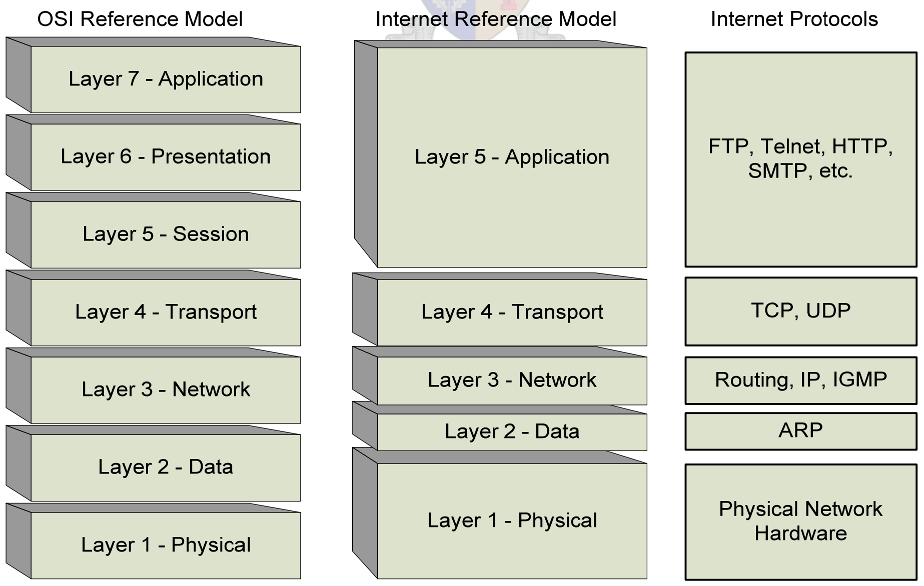


Figure 2.1-2: Layers and protocols of the Internet Reference Model

The Internet protocols, indicated in the above illustration, are discussed in more detail elsewhere in this chapter.

Since the Internet Model is not as well defined as the OSI model, variations are found in different publications with regards the lower three layers of the model. There are also various shortcomings in both network models. A typical example, IGMP (used for handling multicasting traffic), operates on top of IP (Network Layer), but does not support the data transports, TCP and UDP, as supposed by the models, according to the layered model. This case is not supported by any of the two models and is usually treated with a different approach.

Despite the grey areas and shortcomings in the two models, however, it is still useful in most instances to use at least one of the models, or elements of the models, to define certain elements of computer networks and related applications. Typical examples are hardware descriptions, for example network switch classifications, directly related to OSI Layers (Layer 1 Switch, Layer 2 Switch, etc.), or application programming interfaces (APIs), usually related to the Internet Model, providing access directly to the Transport Layer (Inter-application communication utilizing TCP or UDP). [2] [4]

The network model, or elements, best suited to the relevant application and protocols utilized should therefore be used in the project definition. The relevance of the models to the development of the mobile streaming application becomes clearer when considering the relevant protocol issues discussed in subsequent paragraphs.

2.2 <u>Mobile Data Network Considerations</u>

Initially, Cellphones were primarily used for voice communication and the networks were designed with this in mind. In recent times however, Cellular communication networks are becoming more and more oriented towards the delivery of high-speed packet data, especially enabling the delivery of Internet content to mobile phones. As the quality of the data services available on the networks improve, so does the range of applications that it can be used for.

Although the latest GSM family of data networks supports almost all functions of computer networks, there are some limitations that must be considered. Some of the major limitations include limitations of the device that accesses the Internet via the data network, as well as the performance and reliability of the networks.

Mobile phones have advanced in such a way that most of the services provided in the Internet can be accessed, but there are limitations in what the devices can do with the available services. High speed data transfer on the phones is very battery intensive, limiting the time that can be spent online. Similarly, the processing power and embedded software support on the phones limit the extent to which content can be accessed, processed or displayed to the user. These considerations play a major role in the design of the mobile streaming application, described in this document. The relevant issues, regarding the software capabilities of mobile phones, are discussed in more detail in Section 2.6.

2.2.1 GSM Data Technologies

Initially data on the cellular networks was primarily used to send network service messages between devices and Short Message Service (SMS) messages between mobile users. In more recent times, cellular operators' data networks are capable of delivering almost all services available on the Internet, including file transfers, email, browsing and multimedia streaming applications. The ability of the cellular data networks to enable this relies on the fact that the networks are based on the Internet Protocol (IP), which was briefly discussed in the previous chapter. [5] [6]

The cellular networks available in South Africa are all based on the Global System for Mobile (GSM) Communications standard. The GSM family of communication technologies includes voice- and various data services. The data services are provided seamlessly to the end-user, by utilizing the IP protocol and therefore the underlying network structure is not of critical importance to the user, since the user will often be unaware which type of communication is used, or when the type of service is changed, even while accessing the data network. The quality of the service, measured by performance and reliability does play a big role, however, especially when considered in the design of applications that are dependant on the data networks.

The most common GSM data technologies that have evolved, are the following [5]:

- General Packet Radio Service (GPRS)
- Enhanced Data Rates for GSM Evolution (EDGE)
- Universal Mobile Telecommunications System (UMTS)
- High Speed Download Packet Access (HSDPA)

All of the abovementioned services offer IP-based connections with access to almost all content on the Internet. Therefore, the basic measurement criteria as used for normal IP networks can also be used to define the performance of each of the technologies. Since the technologies are seamlessly deployed, with overlapping coverage areas, it is important that the mobile streaming application can operate successfully on all of the abovementioned services and that the difference in performance of the networks be taken into account, if an acceptable user experience is to be achieved.

The following paragraphs give an overview of the performance of the mentioned data networks.

2.2.2 Network Performance

When using an IP communications network, the performance parameters described in the following section, directly influence the user experience. In the design of the streaming application, these parameters must be taken into account to ensure that the user experience is acceptable across the whole array of available GSM networks.

The parameters can briefly be described as follows:

2.2.2.1 Latency

Network latency can be defined as the time required for the transfer of an empty message, from one device, to another device on the network. Network latency is typically measured in milliseconds (ms).

In video streaming applications, the Total latency is defined by the time it takes for the video image to enter a video capture device, to convert and encode the image, transfer it over the communications network and to receive, decode and display the image on the receiving side. Therefore the Total latency includes the Network latency, as well as any additional latency due to processing, buffering or latency of the output device.

2.2.2.2 Bandwidth

The bandwidth of a network is usually one of the most important performance parameters to influence the user experience. The bandwidth of a network determines the amount of data that can be transmitted / downloaded over the network within a given amount of time. The higher the bandwidth, the better the user experience. The bandwidth of GSM data networks is usually measured in kilobits per second (kbps).

The effect of bandwidth and relevance to the design, are discussed in more detail in Section 2.5.

2.2.2.3 Availability

The availability of a network can be defined as the ability of the network to provide a permanent connection, or as the ability of the network to provide the necessary network link, when a connection is required.

The GSM family of data networks does provide an always-on type of capability, but as stated before, the limitations of the cellphones, in terms of battery life, limits the use of this ability to more fixed installations, to a great extent. Other issues, for example signal coverage and contention ratios also play a role in the availability of the cellular data network.

The available GSM/IP-based data networks are compared, in terms of the above performance parameters, as follows:

2.2.2.4 GPRS

GPRS is the most common of the GSM family and deployed by the most Cellular operators worldwide. The coverage of the GPRS network is very similar to the coverage of the related GSM voice services. It is therefore usually assumed that at least GPRS data services are available, where GSM voice networks are available. This assumption holds true for the two major South African Cellular service providers.

The GPRS services share timeslots on the GSM network, with the voice service. The voice service receives a greater priority in the network and therefore the data service is a non-guaranteed service. This implies that, when a GSM base station is busy, the GPRS data throughput will drop significantly, or even be temporarily unavailable. Although a theoretical download speed of 115 kbps is possible, this limit is seldom reached and a figure in the 30 kbps range is more common. In peak times, however, even lower throughput is sometimes experienced, with intermitted periods of no throughput whatsoever.

The latency of GPRS networks is typically in the 600 ms to 1000 ms range. Since the service level and availability is not guaranteed, the user experience with bandwidth intensive applications, for example streaming media, is usually very poor when using the GPRS service. [5] [6]

2.2.2.5 EDGE

EDGE is based on GPRS, but features an enhanced radio interface and transmission techniques, to ensure better radio spectrum utilization. Although the throughput of an EDGE connection declines more rapidly with distance from the base station, than is the case with GPRS, it outperforms GPRS in terms of average throughput.

The theoretical maximum throughput of EDGE is 473.6 kbps. The peak download rates of most EDGE-enabled devices, is 200 kbps, but typical average download speeds of around 100 kbps are mentioned in various technical evaluations. File download tests however suggest that 80 kbps region is more realistic.

The latency of the EDGE enabled network is in the 600 ms region. The enhanced radio communications interface provides for a more reliable connection than GPRS, but the level of service is still not guaranteed. The service can provide relatively good low-bandwith streaming, but in peak times and when not close to the base station, the user experience is comparable to that of GPRS. [5] [8]

2.2.2.6 3G (UMTS)

UMTS is a 3rd generation network, which provides better spectral efficiency, to enable simultaneous voice and data transfer. The UMTS network is based on a sophisticated Quality of Service (QoS) architecture, which makes it very suitable for the transmission of Voice over IP (VoIP), music streaming and video streaming applications.

The speed of a 3G network is also far superior to that of the older GSM networks, allowing theoretical limits of more than 2 Mbps. Most cellular handsets have a 354 kbps maximum limit however, with approximately 220 kbps to 320 kbps average throughput.

The latency of the 3G network is also relatively low, at about 100 ms. Despite the fact that the 3G networks are based on a QoS architecture, the availability of the data service is not guaranteed and the coverage areas are limited, compared to GPRS and EDGE.
The user experience with streaming media, especially audio is relatively good. The 3G networks are however not suited for high resolution video streaming applications. [5] [7]

2.2.2.7 HSDPA

A common view is that HSDPA is for UMTS, what EDGE is for GPRS. This is true in a sense, in that various techniques, similar to the methods used to improve the performance of EDGE over GPRS, are used in HSDPA, to achieve higher performance than UMTS.

The performance of HSDPA is significantly higher than that of UMTS and theoretical download speeds of up to 14 Mbps can be achieved. The first HSDPA supporting devices in South Africa are limited to approximately 1.8 Mbps.

With HSDPA, typical average download speeds of 550 kbps to 1100 kbps are achieved. The network latency is in the 100 ms range.

With the high average throughput and low latency, HSDPA is suited for various applications, especially for streaming- and downloading video and audio content to mobile phones, or even PC's connected to the HSDPA device. There are only a few HSDPA supporting handsets currently available.

Table 2.2-1 summarizes the performance and abilities of the data networks discussed.

Type	Peak Download Speed	Average Throughput	Latency	Availability	Typical Applications
GPRS	115 kbps	30 kbps	600 – 1000 ms	Bursty, intermitted Extended coverage area	Internet Browsing, Email, Telemetry
EDGE	473 kbps, usually limited to 200 kbps by the device	80 kbps - 100 kbps	600 ms	Not guaranteed Extended coverage area	Browsing, Email, Limited multimedia
UMTS	2 Mbps, often limited to 354 kbps by the device	220 kbps - 320 kbps	100 ms	Not guaranteed Limited coverage area	Browsing, Email, Limited media streaming
HSDPA	14 Mbps, currently available in 1.8 Mbps	550 kbps - 1100 kbps	100 ms	Not guaranteed Limited coverage area	Browsing, Email, Video & audio streaming,file downloads

Table 2.2-1 Comparison of GSM family of data services

The impact and considerations, regarding the performance and capabilities of the available data technologies discussed, on the design of the mobile streaming application, are discussed in more detail in Chapter 3, Theoretical Design.

2.3 Data Transport Protocols

This section gives an overview of some of the IP protocols commonly used for the transmission of data on the Internet, as well as illustration of how the protocols interact in a video streaming setup.

2.3.1 User Datagram Protocol (UDP)

UDP is one of the core protocols of the Internet Protocol Suite. It is a very basic transport protocol, which provides applications a method of sending messages to other applications, using a minimum of protocol mechanism. UDP operates on top of IP (Network Layer) and functions as an interface between the Application layer and the Network layer.

The UDP protocol adds very little overhead to packets that are transmitted between networked entities. The protocol applies a small amount of error correction on transmitted data packets and adds IP address-related (port) information to the packets. It does not guarantee delivery of the packets, or ensure that the packets are received in the same order that it was sent, however. It is also known as a connectionless protocol, since no handshaking[1] is done between the sending- and receiving end. Each packet is treated independently.

Figure 2.3-1 illustrates the format of a UDP packet [9]

Bits	Description	
0	Source Port (0 – 15)	Destination Port (16 – 31)
32	Length of data (32 - 47	Checksum (48 – 63)
64 . . .	Data (up to 64 kB, total packet size)	

Figure 2.3-1: UDP Packet Structure

Due to the fact that there is no handshaking in UDP communications and the small packet size, as well as the fact that no packet delivery is confirmed, UDP communication happens almost instantaneously. It is therefore the underlying protocol for DNS, by which domain names are translated into IP addresses. This process needs to happen almost instantaneously, when a DNS lookup is required, therefore UDP is used.

UDP is also very well suited to streaming multimedia applications. This is firstly due to the fact that there is no packet re-sends, by the protocol itself. If a video frame packet is lost, for example, it is not re-sent and therefore the connected device can simply receive and display the following packet, without having a big effect on the user experience, if only one or two packets are lost. UDP is therefore a more efficient transport for applications that do not rely on all packets to be delivered, but rather on the most up-to-date packets to be received.

[1] Handshaking - establishing a connection between the two devices, before data is being transmitted

Another reason for using UDP in some streaming applications is the ability of UDP to simultaneously send the same data packet to multiple devices. This is called multicasting and is often used when multiple users want to access the same media stream. By using multicasting, multiple instances of the same media stream can be prevented, in order to save bandwidth on the network. For full multicasting to work, however, the underlying network must support the multicasting protocol. The detail of multicasting is beyond the scope of this thesis and thus not discussed in this document. [9] [10] [12]

2.3.2 Transmission Control Protocol (TCP)

TCP is the most frequently used of the core Internet protocols. Most of the popular Internet applications, for example email, secure communication and HTTP, rely on TCP.

TCP provides a virtual, reliable 'pipe-like' connection between two applications, unlike UDP, which is connectionless. Therefore, when the TCP connection is established, there is a handshake between the two devices, before data is transmitted. The applications stay connected via the TCP link, for a predetermined amount of time, or until the link is broken by the application.

Below the TCP layer, there is almost no assurance for reliability of data transmitted across the network. TCP provides the application layer with a very reliable protocol layer that assures the transmitting application, that packets sent over the TCP link will be received by the receiving application, in the same order that it was sent.

Apart from providing a reliable connection, the TCP protocol provides some levels of security, as well as congestion control. The further details of these functions can be found in RFC: 793 – Transmission Control Protocol

In order to provide the type of reliability and security, the TCP protocol adds significantly more overhead to the data packets, than UDP. The handshake also results in a longer time to connect, than is the case with UDP, where the packets are immediately transmitted. For each TCP data packet that is transmitted, there is also an acknowledgement from the receiving side, adding to additional traffic on the network.

The TCP packet structure is as follows: [11]

Bits	0–3	4–9	10–15	16–31
0	Source Port			Destination Port
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Reserved	Control Bits	Window
128	Checksum			Urgent Pointer
160	Options (optional)			
160/192…	Data			

Figure 2.3-2: TCP Packet Structure

Unlike UDP, TCP is usually not used for streaming applications, mostly because of the congestion control mechanism implemented by the TCP protocol. When packets are lost, the window size of the TCP stack is reduced and depending on the network response to the reduced window size, the data throughput is reduced. This can cause interruptions in the video stream, especially when high bandwidth streaming is required. [11] [12]

2.4 Application Layer Protocols

This section gives an overview of some of the Application Layer protocols that are related to the development of the streaming video application.

The application layer protocols are a direct interface between the user application and the Transport layer, or it can be incorporated in the application itself, directly making use of the underlying Transport Layer.

Table 2.4-1 shows the typical uses for TCP and UDP: [12]

Application	Application-layer protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	typically UDP
Streaming multimedia	Proprietary/ RTP/ RTSP	typically UDP
Internet telephony	Proprietary	typically UDP
Network Management	SNMP	typically UDP
Routing Protocol	RIP	typically UDP
Name Translation	DNS	typically UDP

Table 2.4-1: Typical applications for UDP and TCP transport layer

2.4.1 Real-Time Transfer Protocol (RTP)

RTP was developed for the real-time streaming of audio and video data over an IP network. Most modern streaming applications are built on top of RTP, since RTP provides a real-time interface with the Transport network layer. The protocol also supports multicasting, which makes it the protocol of choice, when streaming to multiple destinations.

RTP can be viewed as a derivative of UDP, since it is normally used on top of UDP, adding functionality related to real-time streaming applications. In the most basic implementation of RTP, a timestamp and sequence number is added to the UDP packet header. It does not make sense to use RTP on top of TCP, since TCP already handles time stamping and sequencing.

An RTP packet is constructed as follows:

Bits	0	1	2	3	4-7	8	9 - 15	16 - 31
0	V		P	X	CC	M	PT	Sequence Number
32	Timestamp							
64	Synchronization Source Identifier							
96	Contributing Source Identifiers							
128	...							
160 . .	Payload Data							Padding

Figure 2.4-1: RTP Packet Structure

The details of the different fields in the RTP packet can be fount in the *RFC 1889: Real-time Transport Protocol.*

Although RTP provides a facility to the layers above it, to assist in organizing packets that are received out of order, and to optimize quality of service, RTP does not provide these functions by itself. The layers above RTP must use the sequence numbers provided by RTP, in order to reconstruct a packet sequence, or to determine the position of a packet in a sequence. RTP is therefore sometimes described as an incomplete protocol framework.

In many cases, RTP is not implemented as a separate layer, but rather incorporated into the application. Therefore, the application can decide if packets need to be re-transmitted, restrict the sequence, when packets are received out of sequence and perform quality adjustments on the streaming media, according to the performance of the network.

RTP is usually used in conjunction with the Real-time Streaming protocol (RTSP), as well as the Real-time Control protocol (RTCP). The reader is referred to the *RFC 2326* and *RFC 3605* for further reading regarding the details of the complementary protocols.

When different protocols are used together in a network setup to perform a certain application, the collection of protocols is called a protocol stack. The protocol stack for a typical streaming application built on RTP, can be illustrated as in Figure 2.4-2: [13] [14]

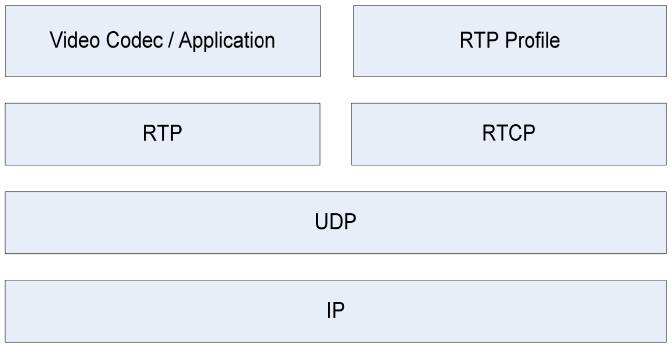


Figure 2.4-2: RTP-based video streaming protocol stack

2.4.2 Hypertext Transfer Protocol (HTTP)

HTTP is the protocol that is most often used on the Internet, to view data, initiate file transfers, start media sessions, etc. In many cases where another protocol is used, for example video streaming over RTP, the first connection that the user makes with the media provider is via HTTP. Although HTTP is a very simple protocol to interpret, it is a very powerful protocol with almost unlimited possibilities.

HTTP is usually used on top the TCP layer. Therefore, sequencing, delivery and performance issues are handled by the TCP layer and HTTP focuses on the delivery of requests to the content provider and the provision of the received responses to the application / user. The HTTP protocol is therefore based on a Request – Response principle.

The chunks of data that are transferred by the HTTP protocol are called resources, which can be identified by a URL (Uniform Resource Locator). The URL contains information on the address, TCP port and sometimes an identifier, by which the resource is located. The address can take the form of a domain name, or an IP address.

Basically any type of data can be transferred across a network, using the HTTP protocol, but since the protocol is based on TCP and a request-response principle, there are some limitations in terms of performance. The performance of the HTTP protocol is similar to that of TCP, but with additional packet overhead. Furthermore, the way in which the HTTP protocol is implemented on most mobile phones, requires a new connection to be made for each HTTP request-response cycle. Therefore, additional latency is introduced for each HTTP communications cycle. [15]

2.5 <u>Video Streaming</u>

This section gives an overview of the basics of video compression and video streaming, as well as an overview of some of the more common video streaming algorithms, compatible with cellphones.

2.5.1 Video Streaming – an Overview

Video streaming can be defined as the transmission of video images from one location on a data network, called the video server, to another location, called the client, without transmitting a single video file. Thus, the video frames can be viewed at the client as it arrives, before all the data has been transmitted. In the case of real-time streaming, for example a web camera, a steady stream is sent from the server to the client in real time, as long as a connection exists between the server and client.

In order for real-time video streaming to work properly, it is essential that the throughput of the data network, which connects the video server and client, is sufficient to transmit all frames at a chosen frame rate, without dropping frames. Frames are dropped when the inter-frame interval (frame period) is shorter than the time it takes for one frame to be transmitted to- and displayed on the client side.

The inter-frame interval t_i equals:

$$t_i = \frac{1}{framerate}$$

The time t_x that it takes for one frame to be transmitted from the server to the client, can be expressed as:

$$t_x = \frac{framesize}{B_{eff}} \quad ,$$

where B_{eff} equals the effective throughput of the communications channel.

Therefore, if $t_x < t_i$,

$$B_{eff} > framesize \times framerate$$

A typical resolution of a modern cellular phone screen is 352 x 416 pixels. Therefore, a CIF (352 x 288 pixels) video frame can be displayed on the screen, without requiring stretching or shrinking of the frame. The total framesize, in bits, of an uncompressed, 24-bit depth color, CIF resolution image can be calculated as follows:

$$framesize = 24 \times 352 \times 288 = 2433024 bits = 304128 Bytes$$

Therefore, at 25 frames per second, the required throughput is:

$$B_{eff} > 25 \times 2433024 = 60825600 bps = 59400 kbps \approx 58 Mbps$$

Thus, the required throughput is approximately 58 Mbps, which is clearly out of range of a normal GPRS, EDGE or 3G internet connection, as well as the amount of RAM available on the majority of cellphones. Even at 1 frame per second, more than 2 Mpbs throughput is required.

In order to transmit the captured frames from the server to the client (cellular phone) over the cellular network, the frames must be compressed, in order to sufficiently reduce the size.

2.5.2 Video Compression

In order to compress the video frames, redundant data is removed from the frames, in such a way as to minimize the visual effect of removing the redundant data. If the redundant data is removed in such a way that there is no reduction in the visual quality of the frames, it is called lossless compression. Although lossless compression does reduce frame sizes, it is normally not enough to transmit the video over a low bandwidth network.

In most cases, lossy compression techniques are used. There are various ways of implementing lossy compression without much loss in the perceived quality of the video stream. These techniques can utilize different image compression algorithms, adaptive compression ratios, reduced framerates, or a combination of various techniques.

The video compression is performed by a video encoder. In order to view compressed video, a decoder is needed. The encoder and decoder must use the same video compression scheme, in order to work. The combination of encoder/decoder within a compression scheme is called a codec, derived from the combination of en***cod***er/***dec***oder. There are several thousands of codecs available on the internet.

Unfortunately, only some codecs are readily available on cellphones compatible with multimedia. Furthermore, not all video codecs support streaming of the encoded content. Therefore, it is necessary to choose carefully when selecting a codec used to stream video to a cellphone. [16]

2.5.3 Video Streaming to Mobile Phones

This section gives an overview of the most common video streaming algorithms. The different algorithms have different advantages and disadvantages in terms of the specific network configuration and applications that it can be used for, as well as compatibilities with certain devices.

The 3rd Generation Partner Project (3GPP) specification covers all the GSM related specifications and protocols and also defines a certain format for streaming media to cellphones. The 3GPP streaming format for mobile phones specifies the following protocol stack (Figure 2.5-1):

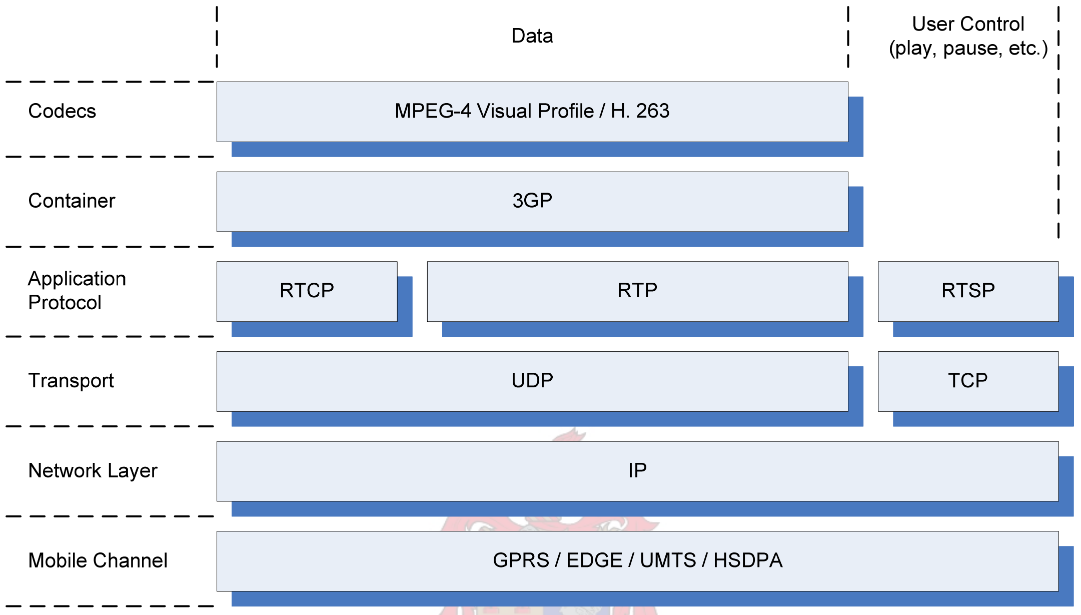


Figure 2.5-1: 3GP Protocol stack

From the protocol stack illustration, the various network- and protocol related issues discussed thus far, become clearer and it can be seen how the basic network model and protocols function in the protocol stack.

In order for a cellphone to be able to decode and view a video stream, it must support the encoding format of the video. There are only few standards widely supported by cellphones, although several others claim to support the 3GPP streaming format.

The following table (Table 2.5-1) shows a list of streaming video technology vendors, as well as supported cellphone manufacturers: [17] [18]

Provider:	Apple	Microsoft	PacketVideo	RealNetworks
Server Operating System	OS X, Unix	Windows Server	Unix	Unix, Windows
Encoder	Quicktime	Windows Media Encoder	PVAuthor	RealProducer
Player Operating System	not defined	Smartphone OS, Windows Mobile 2nd Ed.	Smartphone OS, Symbian	Symbian, PPC Phone Ed.
Streaming Video	MPEG-4, H.263	WMV	MPEG-4, H.263	RealVideo, MPEG-4, H.263
Phone vendor partners	..	Samsung, HTC, Motorola	Sony-Ericsson	Nokia, Siemens, Sony-Ericsson

Table 2.5-1: 3GPP Streaming media technology providers

From the table, it can be seen that the two major supported streaming formats, for mobile phones, are MPEG-4 and H.263. The following paragraphs give an overview of the streaming formats.

2.5.3.1 MPEG-4

The Moving Picture Experts Group (MPEG) is an ISO / IEC working group, which was established to define standards for digital video and audio formats. Various versions of MPEG were developed, each with a different bit-rate or application in mind. [16]

MPEG-4 was developed to enable the encoding of rich multimedia content, extending beyond video and audio and also includes vector graphics and similar content. MPEG-4 is based on object-based compression, in which individual objects in a scene are tracked separately and compressed together, to result in very efficient compression. Data rates supported by MPEG-4 range from 10 kbps to 1,000,000 kbps, which makes it ideal for almost any type of video application. [19] [20]

Although the standard is included in the 3GPP specification, it is an optional standard and thus a 3GPP streaming enabled phone is not required to include support for MPEG-4.

MPEG-4 is a proprietary patented technology, which means that a license fee should be paid to patent holders, if the technology is used in software. There is unfortunately no easy way to find out which patents are applicable to a certain application using MPEG-4.

Since there are some outstanding issues regarding licensing and the fact that it is not a required 3GPP codec, the use of MPEG-4 should not be considered the first choice in the development of a mobile streaming application. [21]

2.5.3.2 H.263

The International Telecommunications Union (ITU) developed the H.263 standard with low bit rate applications, especially video conferencing, in mind. Other uses of the H.263 standard typically include desktop and room-based conferencing, video over the Internet and over telephone lines, surveillance and monitoring, telemedicine (medical consultation and diagnosis at a distance) and computer-based training and education.

Apart from the above uses, H.263 is also required for 3GPP. Thus, all 3GPP supported mobile phones must support the H.263 standard. The standard describes the requirements for the encoder and decoder, to conform, but does not describe the coders itself. The standard includes methods for bandwidth control and synchronization of audio and video, which makes it ideal for real-time applications, with low delay.

H.263 is very processor intensive. H.263 integrated circuits are available to use in conjunction with normal hardware in devices, in order to take the burden of the decoding / encoding from the main processor.

Like MPEG-4, there is also a license fee applicable to H.263, but the fact that it is a required standard for 3GPP, makes it more attractive than MPEG-4. [19] [22]

2.6 Cellular Phone Application Development

Despite the fact that codecs are readily available for use by cellphones, the implementation of a streaming application, with a user-friendly interface requires the development of a suitable user application, compatible with the majority of cellphones.

This section gives an overview of the application development environment available for mobile phones, as well as related issues to consider regarding the development of the streaming video application. Since the majority of mobile phones that do allow 3rd party applications to be installed and used, use the Java technology, the rest of this discussion focuses on the application environment that is provided by the Java framework.

2.6.1 Java Platform

There are currently three major Java Editions in the Java Platform, namely Java Standard Edition (J2SE), Java Enterprise Edition (J2EE) and Java Micro Edition (J2ME). The first two editions are aimed at desktop computers and servers, respectively. The third, the J2ME technology, was developed by Sun Microsystems to provide an environment, in which applications can be created for small devices, with limited memory, display size and processing power.

Figure 2.6-1 gives an overview of the Java Platform structure [24]. The various elements are described in subsequent paragraphs, as related to mobile phones.

J2ME is the Java edition used by Java-enabled cellphones, as can be seen in Figure 2.6-1. The rest of this discussion focuses on the J2ME framework and related frameworks, as applicable to mobile phones.

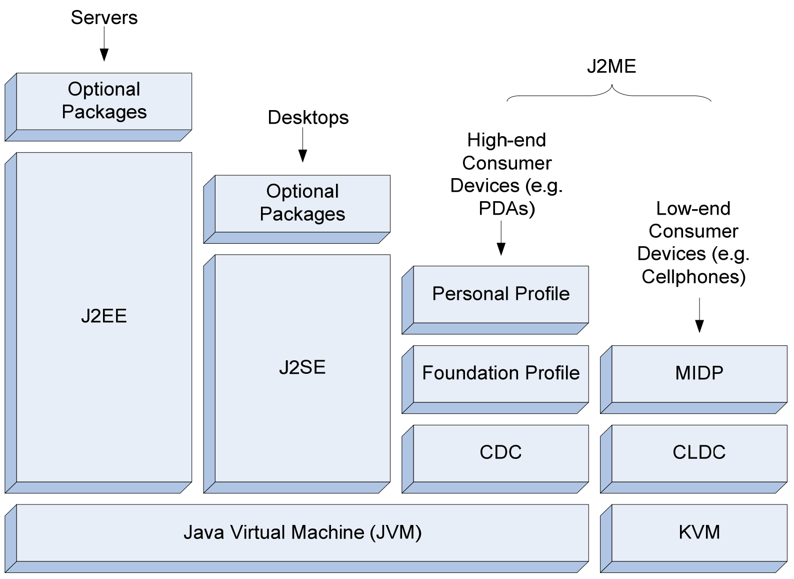


Figure 2.6-1: Java Platform

2.6.2 Java 2 Platform, Micro Edition (J2ME)

J2ME is the most widely supported language for mobile phones, with more than 267 million phones worldwide that are J2ME compatible, in 2004. This figure increased drastically since 2004, since most new phones support Java.

J2ME contains a subset of J2SE, meaning that only some of the Java classes supported by J2SE, are supported by J2ME. Furthermore, J2ME defines two different Configurations, namely the Connected Device Configuration (CDC) and Connected Limited Device Configuration (CLDC). A configuration defines the minimum Java class libraries and the abilities of the Java Virtual Machine (JVM)[2] that must be supported by a device which implements the specific configuration. The Configuration to which a device belongs, depends mostly on the memory size, processing power and network connectivity of the device.

[2] A Java virtual machine (JVM) is an implementation of the Java Virtual Machine Specification, which interprets Java code for a device's processor (or hardware platform) so that it can perform a Java program's instructions. [26]

2.6.3 Connected Device Configuration (CDC)

The connected device configuration is aimed at devices with at least the following specification [25]:

- At least 512 kilobytes ROM (memory) available for the Java runtime environment
- At least 256 kilobytes RAM available (to hold and run applications)
- Network connectivity, possibly persistent and high bandwidth
- Support for the complete Java Virtual Machine specification

The CDC device profile suggests that it is aimed at personal digital assistants (PDAs) and similar high performance personal devices. It does not include normal cellphones.

2.6.4 Connected Limited Device Configuration (CLDC)

The CLDC is a more limited configuration, aimed at devices with lower specification level than CDC. CLDC is a complete subset of CDC. CDC is not a complete subset of J2SE, but contains some of the classes of J2SE, as well as some additional classes. The following illustration (Figure 2.6-2) shows the relationship between J2SE, CDC and CLDC. The significance of this relationship will become clearer in later chapters.

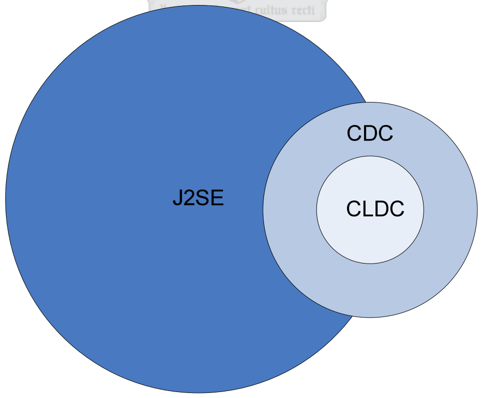


Figure 2.6-2: Relationship between J2SE, CDC and CLDC

CLDC requires the phone to comply with at least the following [25] [23]:

- At least 160 kilobytes ROM available for JVM and CLDC libraries
- At least 32 kilobytes RAM available for runtime memory allocation
- Restricted user interface
- Low power, typically battery powered
- Network connectivity, typically wireless, with high latency and low bandwidth

The phone's operating system must allow the selection and launching of Java applications, as well as the ability for the user to install or uninstall applications.

Cellphones fall into this category and therefore the rest of this discussion focuses on the CLDC class of devices.

2.6.5 Mobile Information Device Profile (MIDP)

On top of the configurations, Sun specifies different profiles. A profile is an extension or supplement to a Configuration, which provides an application programming interface (API) with useful libraries, which a developer can use to develop applications for the particular type of device.

Figure 2.6-3 shows the structure of the various components in a typical device:

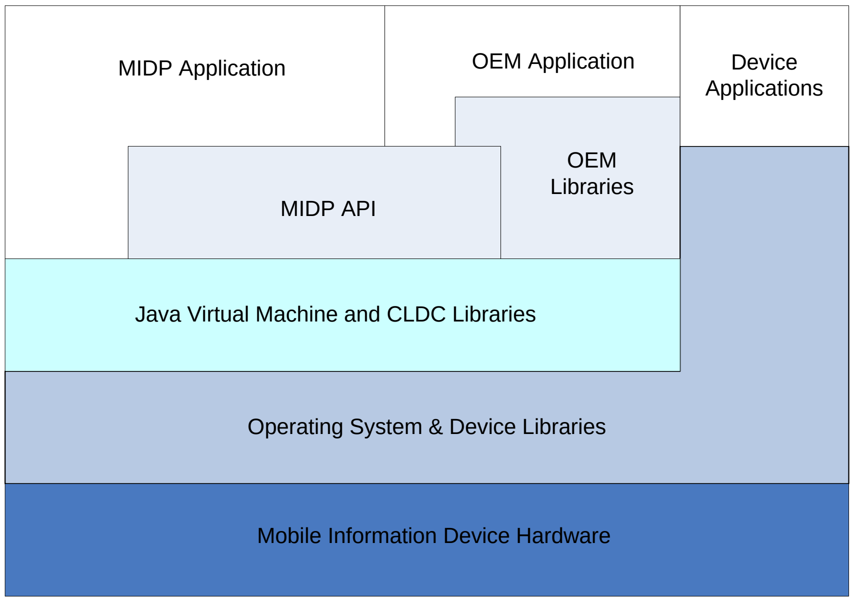

MIDP Application
OEM Application
Device Applications
OEM Libraries
MIDP API
Java Virtual Machine and CLDC Libraries
Operating System & Device Libraries
Mobile Information Device Hardware

Figure 2.6-3: MIDP Application Environment

The top left block in Figure 2.6-3 indicates where third party MIDP applications fit into the device, with both access to the MIDP- and CLDC Java libraries (packages).

2.6.6 MIDP Application Programming Interface

The minimum Java packages available to the application developer in the MIDP environment are the following:

CLDC Core Packages (inherited from J2SE):

```
java.lang.*
java.io.*
java.util.*
```

MIDP Device Specific Packages (part of CLDC packages):

```
java.microedition.io
java.microedition.lcdui
java.microedition.midlet
```

In addition to the required packages, some phone vendors also support some of the optional packages. Since platform independence is required, however, the optional packages are not discussed in further detail. The reader is referred to [23], if more information about the above- and optional packages is required.

In order to ascertain the potential of the MIDP API, for development of a streaming application, it is necessary to be aware of all the restrictions placed on the applications. The following table (Table 2.6-1) gives a summary of the supported functions and requirements of the MIDP API.

Description	MIDP 1.0	MIDP 2.0
Display Size	> 96 x 54	> 96 x 54
Display Depth	1 bit	1 bit
Display Aspect Ratio	1:1	1:1
Input Device	Keyboard or touch screen	Keyboard or touch screen
Memory: Non-Volatile, in addition to CLDC requirement	128 KB min	256 KB min
Memory: Non-Volatile for persistent data	8 KB min	8 KB min
Volatile for Java runtime	32 KB min	128 KB min
Networking: HTTP 1.1	Required	Required
Networking: Secure HTTP connections	Optional	Required
Networking: TCP Sockets	Optional	Optional
Networking: UDP Datagram	Optional	Optional
Networking: Performance	Two-way, wireless, possibly intermittend, limited bandwidth	Two-way, wireless, possibly intermittend, limited bandwidth
Image Format: PNG Support	Required	Required
Image Format: ISO JPEG Support	Optional	Required
Sound (Mobile Media API)	None	Play tones Media optional
Video (Mobile Media API)	None	Optional

Table 2.6-1: Summary of applicable MIDP requirements

As is clear from the table, there are currently two versions of MIDP, namely MIDP 1.0 and MIDP 2.0. The majority of cellphones support at least MIDP 1.0, with CLDC 1.0 It is important to note that MIDP 2.0 is backwards compatible with MIDP 1.0, which enables MIDP 1.0 applications to run on MIDP 2.0 phones. Therefore it is safe to assume that all Java enabled cellphones, compatible with the MIDP profiles, can support applications written according to the MIDP 1.0 specification. It is important to note that the only required network connection included in MIDP is the HTTP connection. Another important point of note is the fact that Video support in the Mobile Media API is not required for MIDP 2.0 and not supported by MIDP 1.0 The relevance of these issues is discussed in more detail in Chapter 3. [27][28]

2.6.7 Deployment of the MIDP Application

The deployment of MIDP applications can be done over a wireless link, or via software installed on a PC, if supported by the phone to be installed on. The phone has a preinstalled Java Application Manager (JAM), which controls the installation, or uninstallation of Java applications on the phone. The JAM is a device-specific application, of which the implementation can differ between different phone manufacturers, or even between different models of the same manufacturer.

The main class of a MIDP application is called a MIDlet. A MIDlet Suite consists of several MIDlets that are packaged together. Each MIDlet application can consist of various Java classes, images and resources. All these files are compressed and packaged into a single file, called a Java Archive File (JAR), before it can be loaded on the phone.

The JAM initiates the MIDlet suite, by using another file that accompanies the JAR, namely a Java Application Descriptor file (JAD). The JAD contains information about the application and allows the JAM to determine compatibility and space requirements, before allowing the MIDlet suite to be initialized.

When initialized, the JAM can start the MIDlet application on the phone and allow it to use all the available CLDC- and MIDP classes compatible with the phone. [27][28]

2.7 <u>Video Server Considerations</u>

Thus far, most of the background provided covers the phone application- and related environments. In order for the application on the phone to be useful, the video content to be used by the application must be provided by some means. The video server, or in the case of this project, the Digital Video Recorder (DVR) is the content provider.

Since the project focuses on the development of a Microsoft Windows PC-based DVR, it is assumed that the server application can be developed to adapt to the limitation of the weakest link in the video streaming setup, namely the mobile application. It is therefore assumed that sufficient processing power and memory is available at the DVR, as well as a sufficient Internet connection, at least capable of providing content to the phone at local ADSL access speeds. ADSL is assumed to be the server side Internet connection, since this is currently the connection that provides the best cost / performance ratio of the available Internet connections in South Africa [29]. A more detailed discussion regarding the DVR specification and requirements is given the subsequent chapters.

2.8 Summary

In this chapter, the various issues that govern the design and capabilities of the streaming application were discussed. An overview was given of the different network models that can be used to describe the communication channels, between the phone and DVR. The capabilities and performance of the applicable GSM data connections were discussed, including an overview of some of the communication- and streaming protocols supported by the networks. Finally, the application environment, as well as capabilities and limitations were highlighted. It was mentioned that the server application (DVR) will be designed according to the (limited) capabilities of the phone application and underlying network infrastructure.

All of the information discussed so far is required in order to determine a realistic approach for designing the streaming application, deployable to the majority of Java-enabled cellphones. The following chapter implements a combination of the characteristics, features and limitations in the theoretical design of the DVR and mobile phone application. Therefore the relevance, of the background and information provided thus far, becomes clearer in the subsequent chapters.

3 Theoretical Design

The objective of this project is to develop a DVR with a streaming video application, which can be used on the majority of Java enabled cellphones. Chapter 2 discussed various issues related to the performance and capabilities of cellphones and the GSM data networks as well as resources available to the developer.

This chapter focuses on the theoretical, or block-diagram design of the DVR and streaming video cellphone application. A high-level design implements the knowledge from Chapter 2, to derive a suitable network model, application characteristics and high-level specification for the DVR and streaming video application.

Firstly, a communications model is derived for the project. Since each respective layer of the model is dependent on the layer below it, it is also limited by the layer(s) below. Therefore, similar to Chapter 2, the discussions in this chapter start at the lowest layer / level, the GSM data network and proceed to the highest level, the user interface.

The streaming video protocol to be used is then discussed and the DVR and cellphone interaction designed up to block diagram level. The preliminary program-flow of the DVR and mobile application is then derived. The sections in this chapter are concluded with functional block diagrams of the DVR, underlying network infrastructure and the streaming video application on the cellphone.

3.1 <u>Network Model</u>

This section describes the derivation of the communications model, on which the rest of this project is based. The network model, available layers- and protocols determined, are used throughout the rest of this document, since all the high-level development is based- and dependent on the available protocols in the network model.

The layers of the model are derived from the distinguishable layers and protocols, as discussed in the previous chapter. These can be summarized as:

- GSM Data connection family – GPRS, EDGE, 3G, HSDPA
- IP Protocol, used on top of all of the above
- TCP and UDP, supported, but not required by MIDP / CLDC
- HTTP and RTSP on TCP and RTP and RTCP on top of UDP
- 3GP with MPEG-4 or H.263 on top or RTP and RTCP
- User application on top of HTTP, TCP, or MPEG-4 and H.263

The following model (Figure 3.1-1) can be used to represent the above summary:

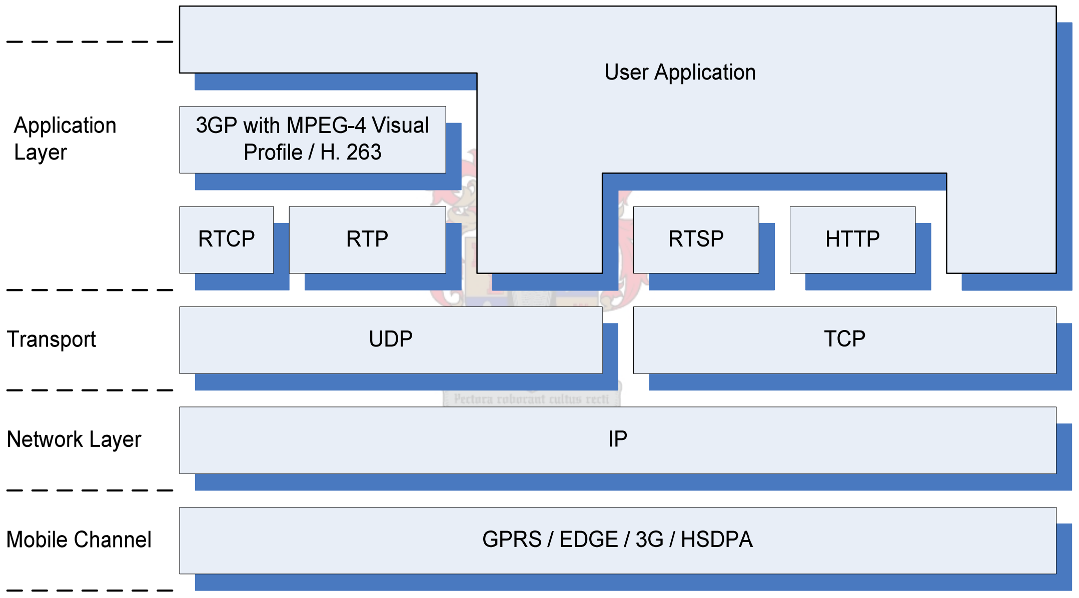


Figure 3.1-1: Network Model for Streaming Application - All inclusive

This model is based on an environment, which includes all CLDC- and MIDP packages and allows the Java application on the phone to have direct access to the Transport layer (TCP and UDP). A real-life, portable application, which is able to run on all versions of the J2ME platform, cannot be based on this model, however.

The following paragraphs discuss the characteristics and phone compatibility of each of the layers of the model, as well as the effect on the higher layers, in order to derive a model that represents the majority of Java enabled phones.

3.1.1 Mobile Channel

The performance of the mobile channel depends on the type of connection that is used. The minimum capability of a Java-enabled cellphone with Internet, is GPRS only. The maximum local capability is currently HSDPA. Therefore, the upper layers must be able to function on any of the GSM data technologies.

The average network throughput, at which the upper layers must be able to function, therefore ranges from GPRS average throughput, namely 30 kbps, up to the HSDPA average, of 550 kbps. As mentioned earlier, the GPRS connection is sometimes intermitted, which the upper layers must also be able to handle.

Furthermore, when a mobile phone moves from an area with a faster type of connection into an area with a slower type of connection (e.g. from HSDPA enabled location to a GPRS-only location), there is an automatic switch of modes by the phone.

The MIDP specification requires the communications packages made available to the application to handle this type of connection (Refer to Table 2.6-1). The layers, below the User Application (Refer to Figure 3.1-1) are included in the specification and thus it is only the implementation of the User Application that must take the above into account. Since the developer does not have any control or direct access to this layer, it is included in the model as is.

3.1.2 Network Layer

All network layer communication is IP-based. The developer does not have direct access to the network layer. Therefore, the network layer can be used as is, in the model.

3.1.3 Transport Layer

The transport layer supports TCP and UDP. However, direct access to the transport layer (TCP sockets or UDP datagrams), is not a requirement of any of the versions of CLDC / MIDP. It is up to the manufacturer to decide if sockets and datagrams are supported by a particular phone model, although MIDP 2.0 recommends that it should be supported. Many of the MIDP 1.0 phones do not give the MIDlet any direct access to the transport layer.

Therefore, again to ensure that the streaming video application is portable to as great a number of handsets as possible, the assumption is made that no direct access is given to the transport layer. Since the majority of new handsets do support sockets and datagrams the ideal setup would be to utilize it if it is available, but not to be dependent on it.

3.1.4 Application Layer

The application layer includes the video streaming protocols, as well as HTTP.

Access to HTTP is a requirement for both MIDP 1.0 and MIDP 2.0. It is therefore safe to assume that an HTTP-based application will work on all Java enabled phones.

The video streaming protocols (MPEG-4, H.263, RTP, RTSP, RTCP) are included in an optional MIDP package, namely the Mobile Media API (MMAPI). The MMAPI is not included in MIDP 1.0. In MIDP 2.0, only the audio capabilities are required, while the video capabilities are optional. [23]

By browsing the websites of the major phone manufacturers, it is clear that currently only the high-end handsets support video streaming as a standard. In order to support video on MIDP 1.0 and low-end MIDP 2.0 handsets, the streaming video functionality of the MMAPI is not available and therefore an alternative video streaming method must be developed and implemented. Due to the limited support for the MMAPI, the protocols related to the MMAPI are, therefore, not included in the network model.

3.1.5 Revised Network Model

The network model can now be refined, to include only the protocols that are available on the majority of devices, spanning the MIDP 1.0 and MIDP 2.0 range of handsets. The revised network model is shown in Figure 3.1-2.

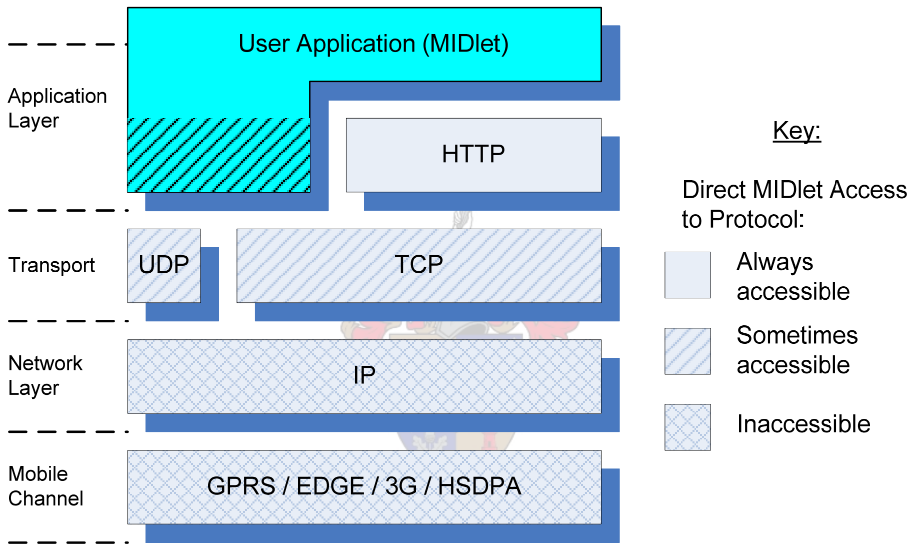


Figure 3.1-2: Revised Network Model

The derived network model has resemblance of both the ISO- and Internet reference models, in which the bottom two layers are combined in the form of the Mobile Channel. The user application is included with the application layer and does not just use an interface to the application layer, since the streaming protocols can be incorporated in the user application. This is discussed in more detail later.

Thus far, the discussions only applied to the mobile application. As mentioned earlier, the DVR design is more flexible and is adapted to comply with the MIDlet requirements. To design the DVR network model, a decision must be taken on the network protocols that will be supported, as well as to how the video streaming will be accomplished, with the limited available resources.

3.1.6 Video Streaming Transport Considerations

By applying the information from the previous section, it is clear that, in order for the mobile application to communicate with the DVR, it can only assume the availability of HTTP connections. This poses a great challenge in the design of the streaming application. In Chapter 2 it was shown that, under normal circumstances, video streaming makes use of UDP and RTP. UDP is not a requirement of MIDP and thus not necessarily available on the phone, only HTTP connections are guaranteed to be available. Support for HTTP connections is therefore mandatory on the DVR. The streaming video application must, therefore, be developed on top of HTTP.

By studying the TCP and HTTP protocol specifications in detail, is becomes clear that it is relatively easy to realize a custom implementation of HTTP, using TCP sockets. This allows the support of both TCP and HTTP, simply by implementing HTTP on top of TCP on the DVR. With both TCP and HTTP supported on the DVR, it makes logical sense that the streaming protocol to be developed, should at least be able to work over HTTP, and optionally also over TCP.

Both TCP Sockets and HTTP are based on the same reliable communication method of communication, which ensures that data packets are received in the same order as it is sent. It is, therefore, safe to assume that a streaming protocol developed for one of the two, should be easily adaptable to work over the other as well, if cleverly designed. This does not apply to UDP, however. It was already mentioned that UDP is better suited for real-time applications and TCP and HTTP are not ideal. There are, however some real issues to consider, before the decision could be taken to also implement UDP as an additional option.

UDP does not ensure correct sequencing of received packets, or provide the data traffic congestion control mechanisms implemented by TCP. Sequencing and bandwidth control is normally handled by a combination of the streaming protocols that operate on top of UDP, namely RTP, RTCP and the video codecs. Although these protocols are supported by the MMAPI, it is also optional. Even when UDP is supported by the handset, there is still no guarantee that the full MMAPI is supported by the handset. These layers can be implemented on top of UDP for phones that do not support it, but this will mean that two parallel streaming mechanisms will have to be developed and implemented, to ensure support for HTTP and optional support for UDP and TCP.

The different scenarios discussed above are better explained by the following flow-diagrams. Figure 3.1-3, represents the cellphone application and Figure 3.1-4, the DVR:

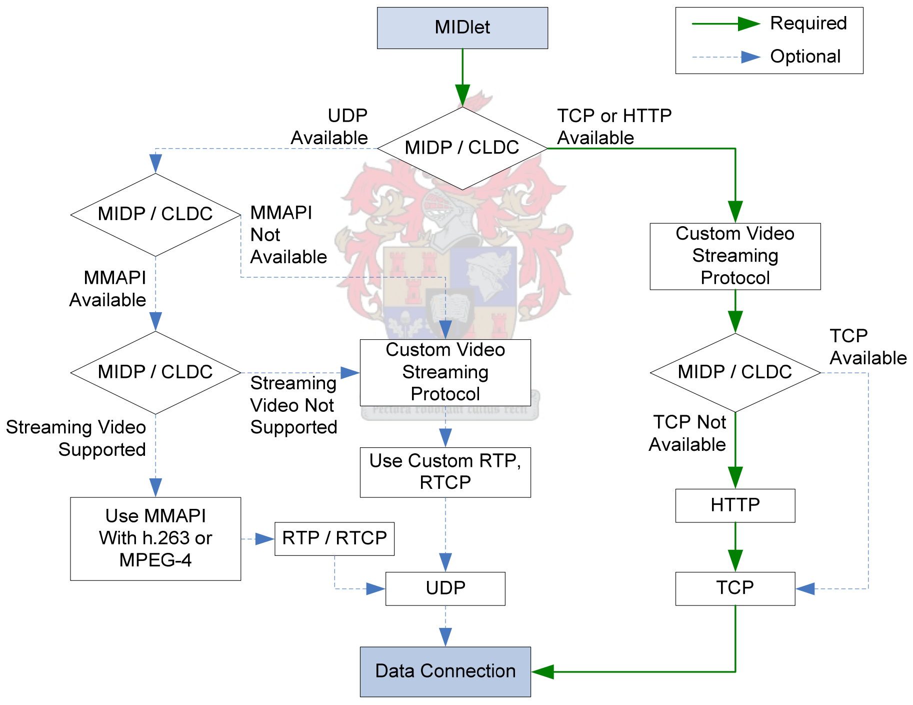


Figure 3.1-3: Flow Diagram for Cellphone Application

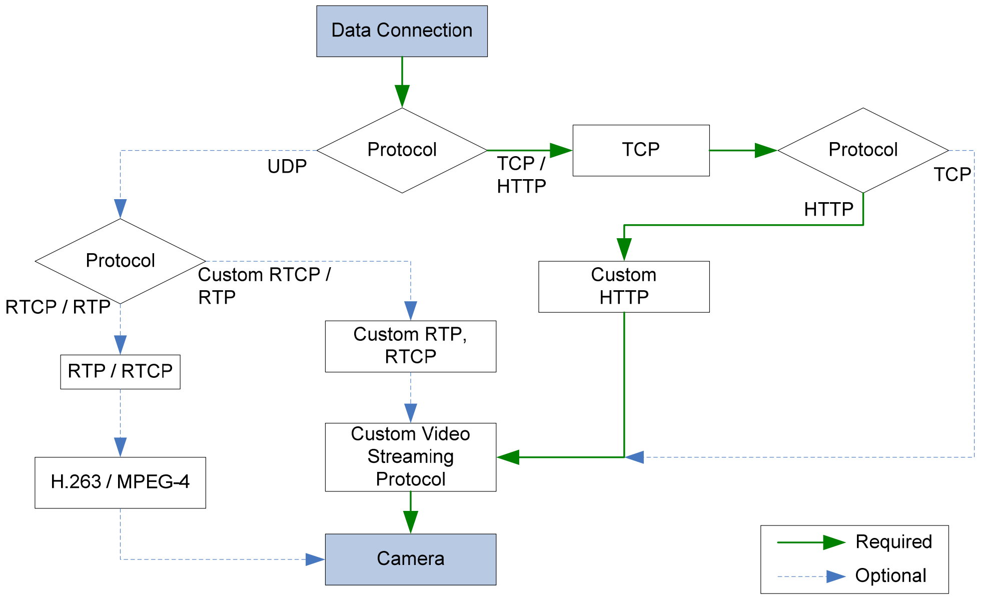


Figure 3.1-4: Flow Diagram for DVR

By studying the flow diagrams, it is clear that in order to make provision to support the possibility that UDP and/or H.263 or MPEG-4 is available on the phone, a lot of additional complexity is added to both the phone application and the DVR. Since only a small number of phones can use this, it makes sense to regard the support for these protocols, on the DVR, also as optional.

It is also clear that TCP can be accommodated in the design, without adding much complexity. Therefore, the rest of this design is based on the development of the DVR, phone application and streaming video protocol with support only for TCP and HTTP connections.

3.1.7 DVR Network Model

By implementing the above decision and Figure 3.1-4, the DVR network model (Figure 3.1-5) is derived.

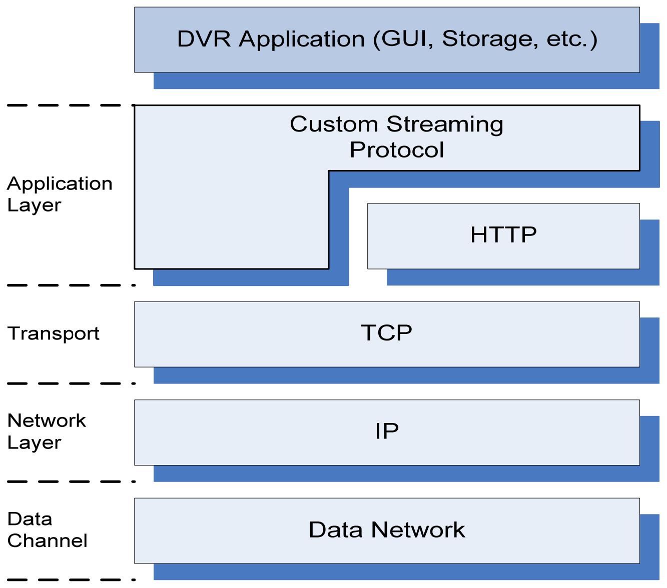


Figure 3.1-5: DVR Network Model

By comparing Figure 3.1-2 and Figure 2.1-1, a functional communications model and protocol stack is derived for this project. This model is shown in Figure 3.1-6.

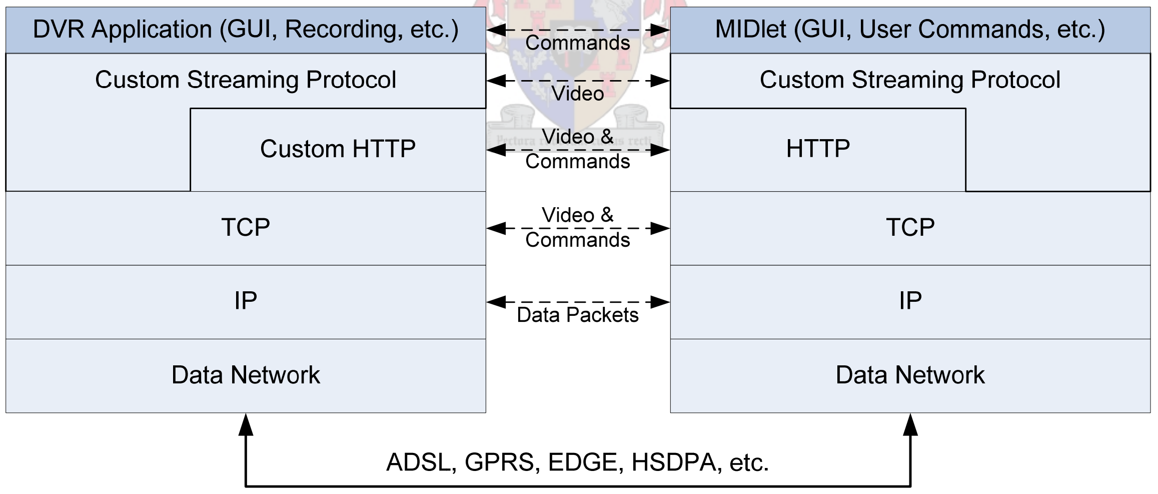

DVR Application (GUI, Recording, etc.)
Custom Streaming Protocol
Custom HTTP
TCP
IP
Data Network
Commands
Video
Video & Commands
Video & Commands
Data Packets
MIDlet (GUI, User Commands, etc.)
Custom Streaming Protocol
HTTP
TCP
IP
Data Network
ADSL, GPRS, EDGE, HSDPA, etc.

Figure 3.1-6: Communications Model & Protocol Stack

The communications model can now be used as the basis for the design of the DVR-to-cellphone streaming video protocol.

3.2 Streaming Protocol:

This section describes the development of a video streaming protocol, between the DVR and cellphone, based on the knowledge from Chapter 2 and the communications model derived in the previous section.

3.2.1 Video Streaming Requirements

The first step in designing the protocol, is to determine a set of video streaming requirements. This set of requirements is based on the project objective, communications model, as well as other considerations discussed thus far.

- The most basic requirement is for high quality video images, to be independent of the data network used. The streaming protocols supported by most cellphones provide video streaming that is suitable for watching a video or for video conferencing. The quality of these protocols, at low bitrates, is however, not good enough to be able to distinguish detail in the background. Some protocols, for example, provide low quality video at low bitrates, in which different objects, for example a person speaking and moving in the video image, can be discerned (sometime called ‘Talking Heads’ video streaming), but with blurry image quality. This is perfectly acceptable for video conferencing, or watching a downloaded video clip, but not for surveillance. With the majority of video streaming protocols, the video image quality decreases, as the bandwidth decreases. [30]

- For surveillance, a different approach is required. A constant video image quality can be maintained, by reducing the framerate, instead of dropping the quality of the video when the bandwidth decreases. By sending all video frames, encoded at the same high quality, to the cellphone and dynamically adjusting the framerate, an optimal video image quality can be maintained, regardless of the network throughput. Such a video streaming protocol would be more suitable for surveillance applications.

- Another requirement for the streaming protocol, is that it must support the ability to switch between cameras, therefore halting the streaming of a particular camera and starting the streaming of another camera. In a conventional video streaming system, this functionality is normally provided by RTSP. Since the assumption is made that it is not available, this must therefore be implemented in the protocol.

- The streaming protocol must also be able to support TCP and HTTP connections, as discussed in the previous sections.

- Another important requirement of the protocol is that the video decoder, which will be implemented fully in software, must be light on processing resources. Most cellphones have limited processing power. The decoder must therefore be designed to use as few processor intensive algorithms as possible, to ensure compatibility with the limited processing power, especially of older phones.

3.2.2 Designing the Streaming Video Protocol

This section describes the design of streaming video protocol, designed according to the requirements discussed in Section 3.2.1.

In order to derive a simple, processor-friendly video streaming algorithm, it is best to start with a very basic approach. Full motion video is a sequence of images, presented at a high enough rate to appear like fluid motion. Digital video can present the images in a number of formats, providing different levels of quality and video compression. Video streaming algorithms, typically compress individual frames as well as implement algorithms between frame sequences, in order to optimize the ratio of compression. This obviously requires more processing power, especially if implemented in software. More memory is also required. More than one image is typically required to decode the video stream, because of the fact that inter-frame compression is used. This requires more memory, to buffer images. Implementing such a memory- and processor intensive algorithm is therefore not ideal, especially for older MIDP 1.0 handsets.

A simpler type of streaming is to only compress each frame, without any compression between different frames. This is typically achieved, by streaming a sequence of compressed JPEG (Joint Photographic Experts Group) images. This method of streaming is called Motion JPEG, or MJPEG. No formal standard for MJPEG exists and therefore there are several flavors of MJPEG used for different applications.

MJPEG usually have a lower compression ratio than other video streaming algorithms, but due to the fact that each frame is individually compressed, the image quality is very good, even at very low frame rates.   This quality makes it especially attractive for low frame rate applications.

Most modern IP cameras support MJPEG streaming over HTTP. This allows access to the cameras and full motion video to be viewed using a standard web browser. [21]

Although the above seems like the ideal type of video streaming to meet the requirements set out in the previous section, a quick look at Table 2.6-1 reveals that JPEG support is not required for MIDP 1.0 devices, but only for MIDP 2.0.   Portable Network Graphics (PNG) are supported by both, however.   PNG images utilize lossless compression and therefore do not have very high compression ratios. Since this is the only image format required to be supported by both MIDP 1.0 and MIDP 2.0, it can be used in similar fashion to MJPEG, in order to stream a sequence of frames (in this case PNG images) from the DVR to the cellphone.   JPEG is then supported as an alternative.

Although further algorithms can be used to improve the compression ratio, it was decided that, in order to keep the load on the phone’s processor as low as possible, the streaming algorithm used in this project will be based on sending a sequence of PNG, or JPEG images from the DVR, to the cellphone application.

The details of the video streaming, including the method for switching cameras while streaming, are discussed further in the relevant sections of Chapter 4.

3.3 Designing the MIDlet

This section describes the theoretical design of the cellphone application. It is assumed that the reader is familiar with normal software programming terminology and the Java[3] programming language.

The MIDlet design requires two distinct components, in order to stream video and control the DVR. The first is the user interface, including graphical user interface (GUI), with the various options for the user to send commands to the DVR, request or terminate video streams, display video and input configuration data, related to the DVR. The second component is the networking component, which is used to communicate with the DVR, to package and transmit commands, or to receive and decode video frames received via the network.

As for any user friendly application, it is essential that the GUI must be usable at all times. This means that all controls of the user interface can be used at all times, without locking. In order to realize this, the MIDlet must be a multi-threaded application, with the main thread handling the GUI and user input, while a second thread handles the communication- and video streaming protocol.

The initial MIDlet design allows the user to input and store the DVR IP address and authentication information, as well as to select-, stop- or start a camera. The high-level program flow diagram of the MIDlet is shown in Figure 3.3-1. The implementation of the two threads to be used is also indicated in the figure.

[3] More information about Java and the Java language syntax can be found at www.sun.com

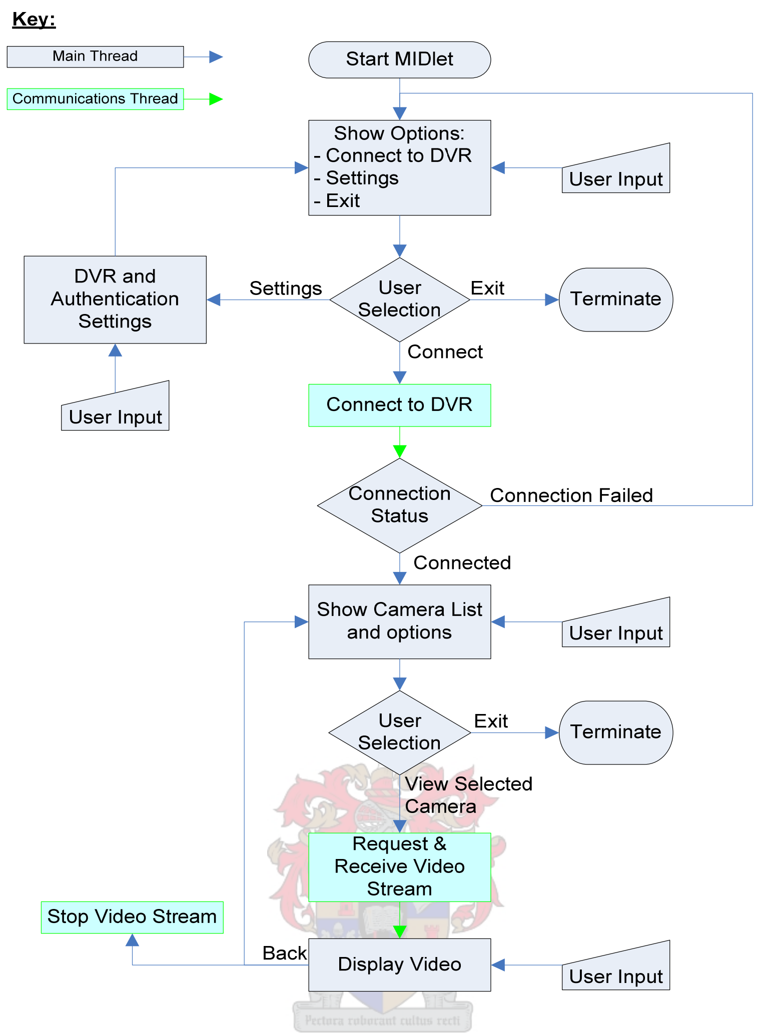


Figure 3.3-1: High-level MIDlet Program Flow

The flow diagram shows that the main thread allows the user firstly to set the username, password and address of the DVR and then to connect to the DVR. When successfully connected, the list of cameras available for viewing is displayed. The user can then select a camera for viewing, or exit the application.

The connection thread handles the connection to the DVR, as well as the video streaming. The frames received by the connection thread are handed to- and displayed by the main thread.

The further program design and MIDlet flow diagrams are discussed in more detail, in Chapter 4.

3.4 Design of the DVR with Video Server for MIDlet Access

This section describes the theoretical design considerations of the DVR with the video server. It is assumed that the reader is familiar with software programming terms and definitions.

3.4.1 Application Development Environment and Programming Language

The first step in the DVR design is to decide on the programming language- and environment to be used. Unlike the case with the phone application, numerous programming languages- and environments are available to develop Windows applications. The various languages have different advantages and disadvantages for different applications. A language suited to the development of the DVR is therefore selected by evaluating the requirements of the DVR application.

The requirements are summarized as follows:

- It is essential that the programming language gives access to devices in the Windows environment, in order to stop-, start- or control camera devices.
- Development time must be short and preferably various third party components must be available to reduce the amount of low-level programming required.
- The language must be well suited to the development of networked applications.
- A simple Integrated Development Environment (IDE) must be available, which enables rapid GUI development, compiling and debugging.
- The end-user application (DVR) must be easy to deploy.

Various programming languages can fulfill the above list of basic requirements, for example Delphi, Java or C++. C++ is a very powerful language, but development time is usually very long, due to the involvedness of the language. Java can be used, with the available media libraries, to develop a media application in a short time, but the capabilities of the media libraries and the performance of Java applications are not adequate to fulfill the above requirements in full.

The language that most closely fits the requirements is Borland Delphi. Delphi was developed by Borland, initially aimed at enabling the rapid production of desktop- and enterprise database applications. It is, however, suited to almost any type of application development.
Various free- and commercial third party components are available for Delphi, to reduce development time and to minimize the amount of low-level programming required. Delphi includes a compiler, full IDE, debugger, compiles very efficiently and the developed applications are easily deployable. Several free third-party components, including networking components, are included Delphi installations, to further reduce development time. [31]

For this project, the availability of a Delphi installation and knowledge of the language also count in favour. Delphi is therefore chosen as the programming language for the DVR, although several other languages could possibly also be used.

3.4.2 DVR High-level Design

In order to do the high-level design of the DVR, the requirements and functions of the DVR are broken up into more manageable elements. The first set of elements consists of the basic operations and functions of the DVR, including the user interface and related functions. The second set of elements consists of the underlying architecture of the software, which enables it to operate and execute the various operations in separate threads and communicate with mobile phone applications.

The following paragraphs highlight the main DVR requirements and required threading architecture, in order to derive the high-level design.

<u>Primary GUI</u>
The first element is the primary DVR GUI, which is the main interface between the user and the software. The primary GUI consists of the following:

- Conventional main program menus, with access to all user input functions

- Video display windows, to enable the user to operate and view cameras, or change camera device- and related settings
- Status window to display DVR operations, information and messages

<u>Secondary GUI:</u>

The secondary GUI consists of windows that are indirect interfaces between the user and the software. The secondary GUI's are accessed via the primary GUI and consist of the following:

- Settings windows for configuring Camera devices, monitoring options, mobile users, alarming options, etc
- Event log, which records or displays a list of all events and messages, with time- and date stamp
- Video review window, to enable the user to view recorded video material

<u>Streaming Video Server:</u>

The video server must enable mobile phones to connect, authenticate and request video streams. The video server must also manage connected mobile devices and stop or start cameras as required for streaming to the mobile phones.

<u>Camera component:</u>

Each camera used by the DVR must have independent settings and a separate viewable screen.

The user must be able to stop / start recording or motion detection for individual cameras.

The camera component must provide the video server with video images, when required.

When motion is detected, the DVR must take appropriate action, by sending an alarm notification, as required by the user.

The DVR underlying architecture consists of the following elements:

- Main program and GUI
- Camera component(s)
- Communication component(s)

Each of the abovementioned components must be able to function simultaneously with the other components. Furthermore, there is possibly more than one instance of the camera component, in the case where more than one camera is used. Similarly, there can be more than one client communicating with the DVR at any given time. All these operations must be able to run simultaneously, without the one blocking one of the others. In order for each camera to do motion detection, recording, or streaming without affecting the operation of other cameras, each camera component must run in a separate thread. Similarly, a separate thread is required for each user connected to the DVR. The realization of the multithreaded DVR architecture is described in more detail in Chapter 4.

The above requirements are used to derive the following flow diagrams for the main- (Figure 3.4-1), connection- (Figure 3.4-2) and camera (Figure 3.4-3) threads.

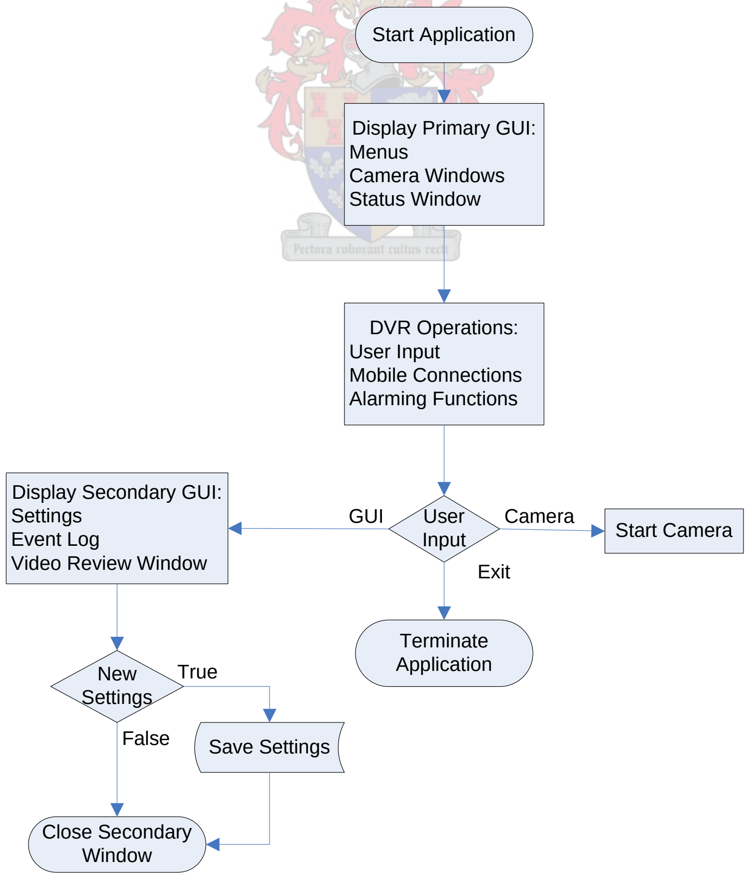

Start Application
Display Primary GUI: Menus Camera Windows Status Window
DVR Operations: User Input Mobile Connections Alarming Functions
User Input
GUI
Camera
Exit
Start Camera
Display Secondary GUI: Settings Event Log Video Review Window
Terminate Application
New Settings
True
False
Save Settings
Close Secondary Window

Figure 3.4-1: Main Thread

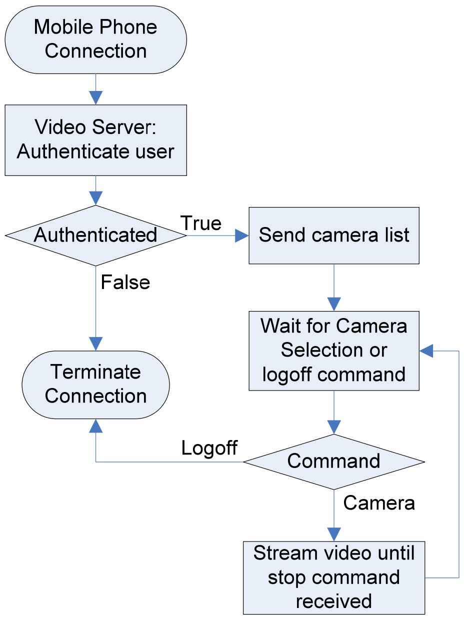

Mobile Phone Connection
Video Server: Authenticate user
Authenticated
True
Send camera list
False
Wait for Camera Selection or logoff command
Terminate Connection
Logoff
Command
Camera
Stream video until stop command received

Figure 3.4-2: Connection Thread

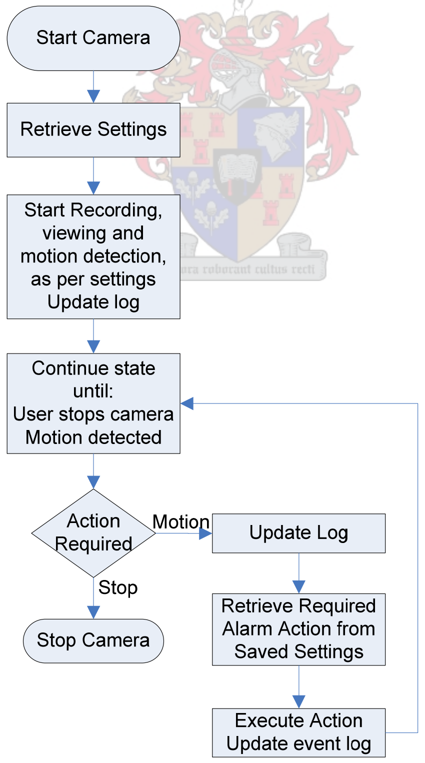

Start Camera
Retrieve Settings
Start Recording, viewing and motion detection, as per settings Update log
Continue state until: User stops camera Motion detected
Action Required
Motion
Update Log
Stop
Retrieve Required Alarm Action from Saved Settings
Stop Camera
Execute Action Update event log

Figure 3.4-3: Camera Thread

3.5 Summary

In this chapter, the network models, theoretical streaming protocol and high level designs of the mobile application and DVR were discussed. The next chapter implements and refines the designs and presents more detailed breakdowns of the components highlighted in this chapter.

4 Implementation

This chapter focuses on the implementation of the theoretical models and background presented in Chapters 2 and 3.

Firstly, the MIDlet implementation is discussed. The DVR implementation is then presented. Detailed program flow diagrams are presented and issues encountered in the implementation, as well as workarounds used are discussed where applicable in the implementations.

4.1 <u>MIDlet implementation</u>

The knowledge and theoretical designs presented in the previous chapters are combined in this section, to develop the application to enable video streaming on the mobile phone. Due to its intuitive user interface, the NetBeans IDE is used for the mobile application development, in conjunction with the J2ME Wireless Toolkit 2.2 from Sun Microsystems.

In Chapter 3, the decision was taken to support both HTTP- and TCP compatible phones. It is theoretically possible for the application to determine if TCP is supported in addition to HTTP, but with the limited memory available on older phones, it is preferable to minimize the amount of redundant code installed on the phone. For the development of the application, two separate applications are developed, to minimize the footprint (size) of the application and to simplify debugging. Therefore, phones that are only HTTP compatible will use the HTTP version of the software and TCP compatible phones, the TCP version. Both versions of the mobile phone application are based on a dual-threaded architecture, as discussed in Section 3.3.

4.1.1 Dual-threaded MIDlet Implementation

This section briefly describes how the required dual threads are implemented.
The MIDlet flow diagram, Figure 3.3-1, is used as the basis for the implementation. In order to implement the dual-threaded application, care is taken, however, to make sure

that no resources are accessed by the same thread at any given time. When both threads try, for example, to access the portion of the screen where the video frame is displayed, possible thread interlocking can occur and cause the application to hang. In order to prevent interlocking, an interface is used in combination with synchronized methods. Therefore the application consists of a main class, which implements the interface, as well as a communications class, in a separate thread, which makes callbacks via the interface, to communicate with the main thread.

The following figure illustrates the implementation of the main thread and interface:

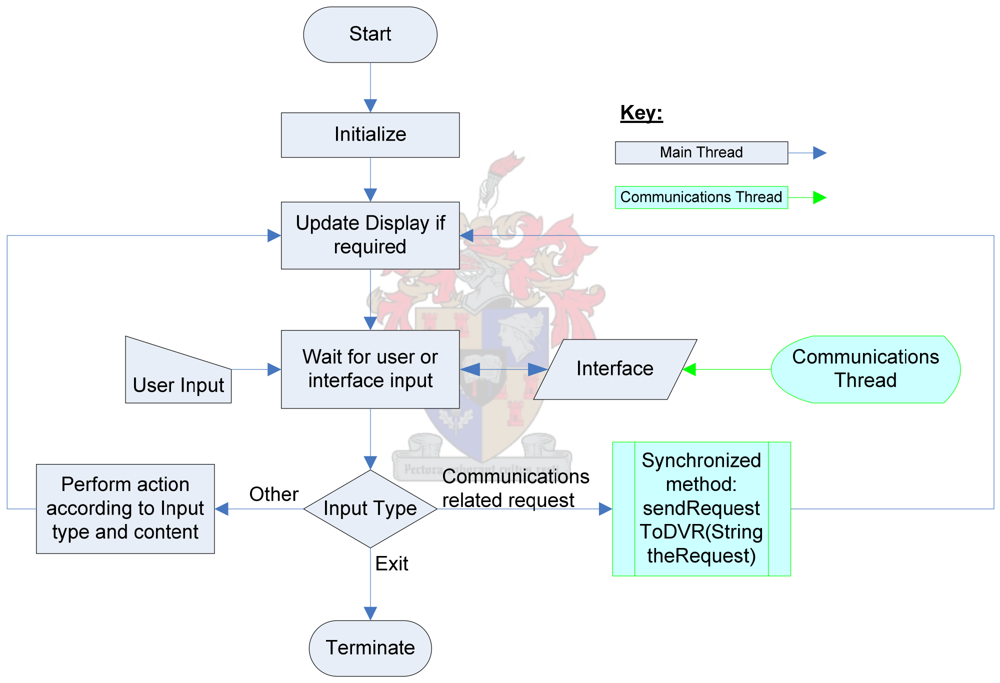

Start
Initialize
Key:
Main Thread
Communications Thread
Update Display if required
User Input
Wait for user or interface input
Interface
Communications Thread
Perform action according to Input type and content
Other
Input Type
Communications related request
Synchronized method: sendRequest ToDVR(String theRequest)
Exit
Terminate

Figure 4.1-1: Main Thread and Interface Implementation

It is important to note from the figure that, after a communications-related request was sent to the DVR via the communications thread, the main thread calls the synchronized method, but does not wait for the communication thread, before it continues to wait for input – both from the user or the communications thread. This enables the user to continue using the GUI, even while the communications thread is communicating with the DVR in the background.

The main thread, in both the TCP and HTTP MIDlet versions, is the same, but the communications threads differ slightly in the two versions. It is therefore discussed separately, in the subsequent sections.

4.1.2 HTTP MIDlet

In order to implement streaming over the HTTP protocol on the cellphone, an understanding, of how the HTTP protocol is implemented on the phones, is required.

Ideally, the phone should make an HTTP connection to the DVR by means of the MIDP HttpConnection object. The phone should then be able to send requests and receive responses, without ever closing the connection between request-response cycles. This would ensure that the latency between transmissions is minimized, by cutting out the connection time between transmissions and enable the phone to send commands and receive video streams, by re-using the same connection. The HTTP 2.0 specification, which is a requirement of MIDP 2.0, allows for the re-use of a connection and can theoretically be used by setting the HTTP header *'Connection: keep-alive'*. Such an HTTP connection is called a persistent HTTP connection. By browsing the manufacturer discussion forums, however, it becomes evident that it is not possible to reuse an HttpConnection instance in MIDP. The HTTP instance changes state, from connecting - to connected - to sending - to receiving, but it cannot return to a previous state. The instance is valid only for one HTTP request-response round-trip and cannot be used again. [32]

4.1.2.1 Connecting to the DVR

In order to communicate with the DVR, the phone must create a new connection with the DVR, every time it wants to send a new command or receive new data. The first implication of this is that the client's (mobile application) authentication state at the DVR is not automatically maintained, because of the fact that the connection is closed. In other words, when the phone sends a connection request, with the correct username and password to the DVR, the DVR sends a response to the phone, which includes confirmation that it is authenticated. After the confirmation is sent, the connection instance cannot be used again and must be closed. Now, when the mobile user selects a camera, a new connection must be made to the phone. The DVR must now 'remember' that the specific phone is already logged in, despite the closed connection.

This can be easily achieved, by sending a unique session ID, in the HTTP header of the initial DVR authentication confirmation response. This session ID is then kept valid by the DVR for a predetermined time. If the phone does not communicate with the DVR again during this time, the session expires and the phone is logged out. The phone application includes the session ID in each subsequent request header, which automatically allows the DVR to recognize it. The session expiration timer is reset every time the phone makes a new request.

4.1.2.2 Receiving the Video Stream

When a video stream is requested by the user, there are two options that can be taken to achieve live streaming. The first is to create a request-response for every frame that is required. This has the obvious disadvantage that the framerate is adversely affected by the connect/disconnect times introduced between frames. The second option is for the server to send a response to the phone, without specifying the content-length header. This is also known as server-push communication. The server can send (push) images to the client, until the client closes the connection. This causes an exception to be thrown, but is easy to handle, since it is a controlled exception. Since the second option only introduces the connect/disconnect delays between requests and not between frames, this option is taken.

4.1.2.3 Streamlining the Communication with the DVR

In order to minimize the amount of requests to be sent between the application and the DVR, the list of available cameras is included in the authentication response, by the DVR. When the client exits the mobile application, a disconnect request is sent to the DVR, to terminate the session and minimize the amount of open sessions at the DVR.

4.1.2.4 HTTP Message Sequence Chart

The following message sequence charts illustrate the communication between the mobile application and the DVR, as described in the previous paragraphs:

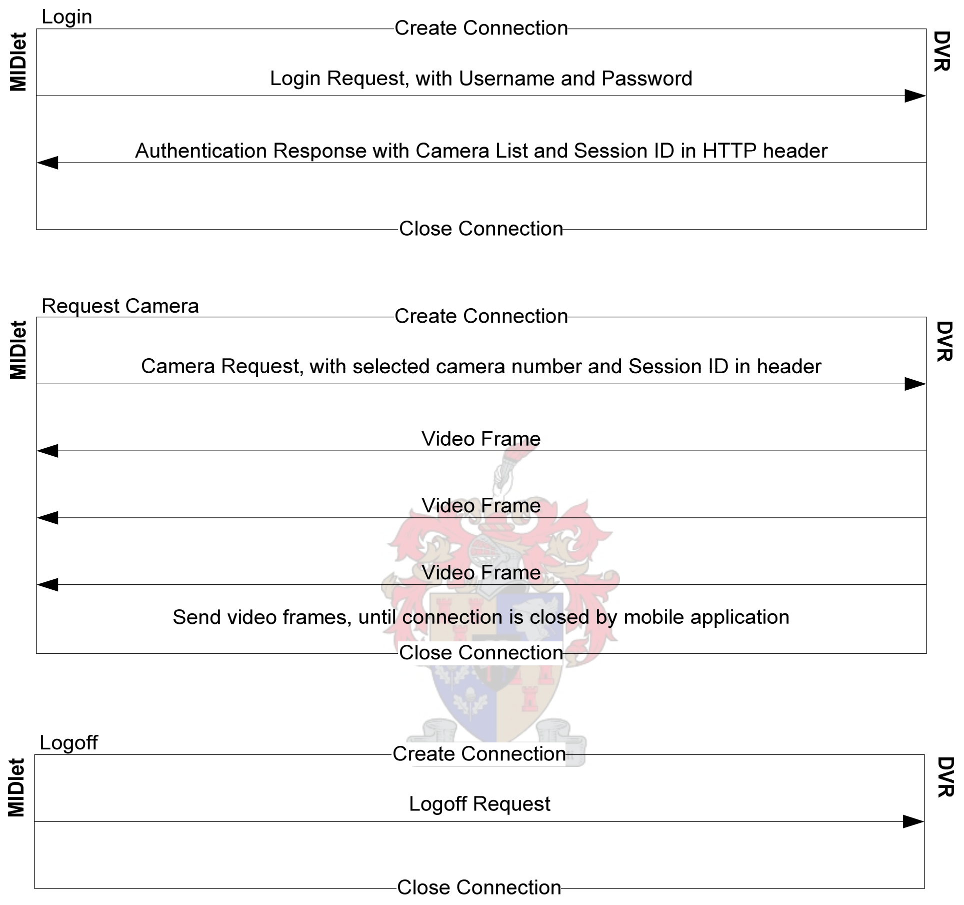

Login
MIDlet
Create Connection
DVR
Login Request, with Username and Password
Authentication Response with Camera List and Session ID in HTTP header
Close Connection
Request Camera
MIDlet
Create Connection
DVR
Camera Request, with selected camera number and Session ID in header
Video Frame
Video Frame
Video Frame
Send video frames, until connection is closed by mobile application
Close Connection
Logoff
MIDlet
Create Connection
DVR
Logoff Request
Close Connection

Figure 4.1-2: HTTP Message Sequence Chart

4.1.2.5 Implementing the HTTP Communication Thread

The HTTP communication thread is implemented in a separate class, given the name ConnectionHTTP. An instance of this class is created by the main thread when the MIDlet is started and the communication thread is then created in the ConnectionHTTP instance. The thread remains in idle mode, until a request is received from the main thread. The main thread sends all requests, by calling the sendRequestToHost method of the ConnectionHTTP instance. The sendRequestToHost arguments include the DVR IP address, the request message to be sent to the DVR and a reference to the main thread interface.

Authentication

When a login request is received from the main thread, the login details are included with the request message. The communications instance now wakes up the communication thread, which opens an HTTP connection to the server address, included in the arguments, transmits the request message with login details and then waits for a response from the DVR. If a successful response is received, the response, with camera list, is sent to the main thread to process, by means of the interface received with the request. In the case of successful authentication, the Session ID is stripped from the HTTP response header and stored by the connection instance. In case an error occurs or no connection or authentication is made, an error message is sent to the main thread.

The flow diagram for the login to the DVR, described above, is shown in Figure 4.1-3:

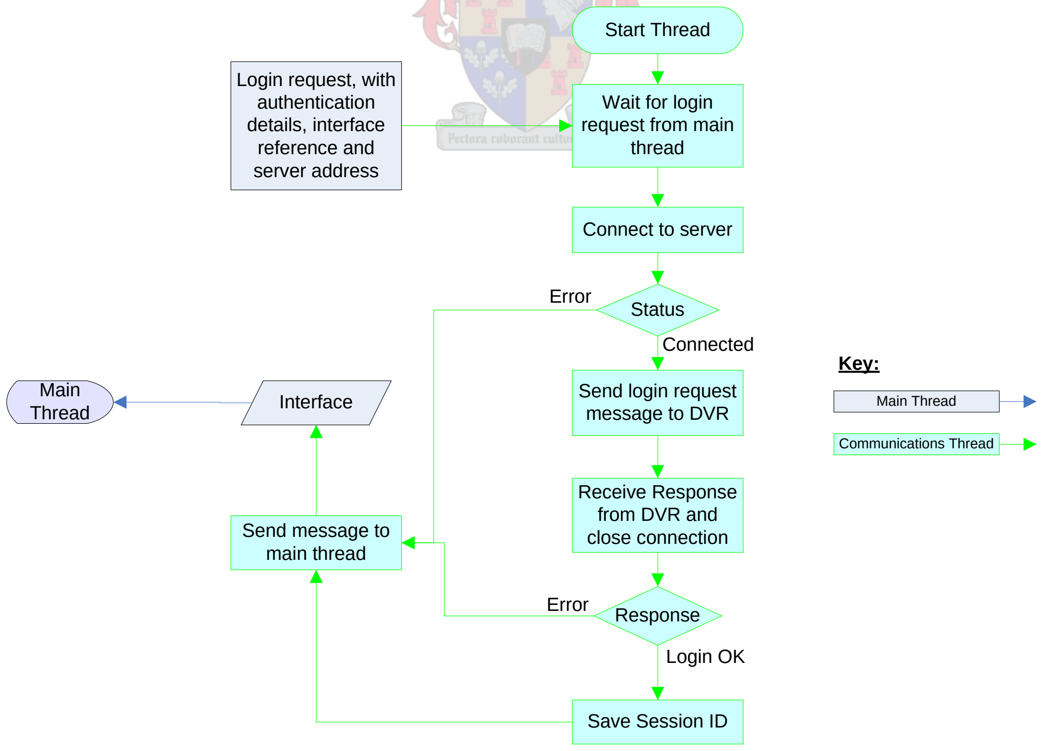

Start Thread
Login request, with authentication details, interface reference and server address
Wait for login request from main thread
Connect to server
Error
Status
Connected
Key:
Main Thread
Interface
Send login request message to DVR
Main Thread
Communications Thread
Receive Response from DVR and close connection
Send message to main thread
Error
Response
Login OK
Save Session ID

Figure 4.1-3: HTTP Login Flow Diagram

Video Streaming

When a stream request is received from the main thread, the communication thread is awoken and forwards the camera number, of which the video stream is requested, to the DVR. The saved session ID is included in the request header. A loop is then entered and images received from the DVR. The DVR sends the size of each image, before each image. This enables the communication thread to only read the correct amount of data from the connection, before creating an image instance with the received data and forwarding the image to the main thread, via the interface, to display. Before the next image is received, the interface's callback, isNewFrameRequired, is called to confirm if streaming is still required. If not required, the connection is closed and the thread returns to the idle state.

The video streaming program flow is shown in Figure 4.1-4

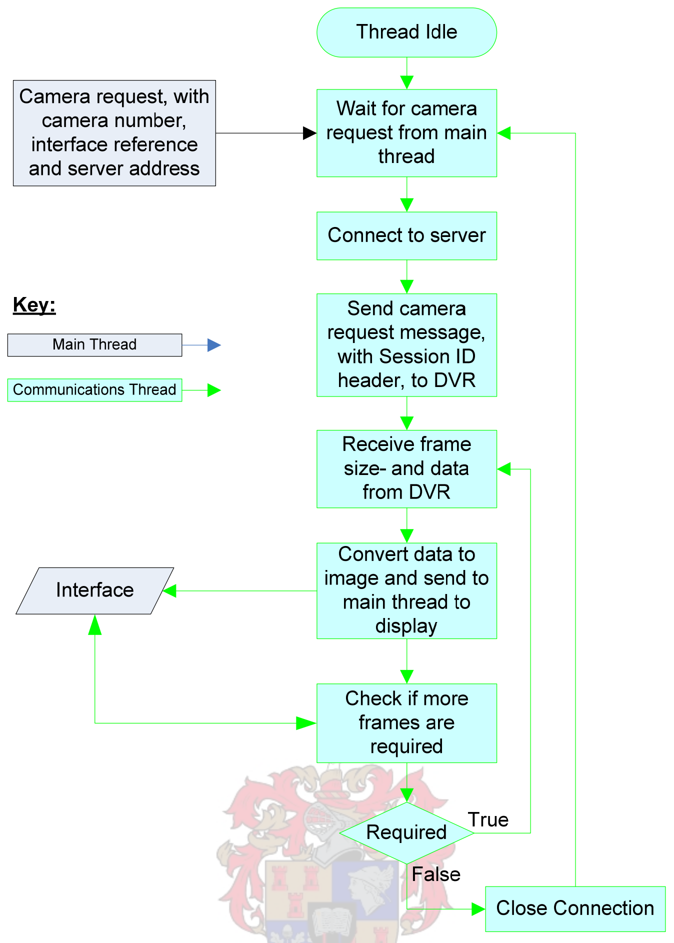

Thread Idle
Camera request, with camera number, interface reference and server address
Wait for camera request from main thread
Connect to server
Key:
Main Thread
Communications Thread
Send camera request message, with Session ID header, to DVR
Receive frame size- and data from DVR
Interface
Convert data to image and send to main thread to display
Check if more frames are required
Required
True
False
Close Connection

Figure 4.1-4: HTTP Streaming Program Flow

Although no error handling is shown in the flow diagram, all errors are handled by the communication thread and reported via the interface, to the main thread.

Interface

In order to enable the required communication between the two threads, the interface is defined with the callbacks as per Table 4.1-1, for use by the communications thread:

Table 4.1-1: CommunicationThreadInterface

interface CommunicationThreadInterface	
void handleImage(Image newImage);	Used for sending image to main thread
void handleResponseMessage(String hostResponse);	Used for sending DVR response message to main thread
void handleErrorMessage(String errorMessage);	Used for error handling
boolean isNewFrameRequired ();	Used to check if new frame is required Returns boolean

4.1.2.6 User Interface

The Graphical User Interface (GUI) is implemented by displaying different forms, according to the application's requirements. The library, javax.microedition.lcdui.*, is used to implement the forms. The camera list is, for example, displayed on a form, by using javax.microedition.lcdui.list. When the list of cameras is received, the list is updated and then displayed.

Various commands can be added to each screen, to enable user input. A Command instance is declared for each command. When a command is selected by the user, the system calls the commandAction method. The action related to the specific command is then executed.

For streaming video, a Canvas object is used to display the video images. A canvas provides smoother update of the display and is typically used by games- and video MIDlets, to reduce flickering when graphics are drawn on the display. A video canvas, developed by Wayne Forrest [33], is used to display the images received from the connection thread. The settings form, which saves the login information and server address in the phone's memory, by the same developer, is also used in the MIDlet.

Due to the way different manufacturers implement the javax.microedition.lcdui library, the GUI looks different on different phones. This is illustrated (Figure 4.1-5 and Figure 4.1-6) using different emulators, to select the 'Menu' command, using the same MIDlet.

Although this limits the ability of the developer to customize a GUI in MIDP applications, cellphone users are generally used to different interfaces on different phones and should therefore be able to adapt quickly, when using the application on different phones.

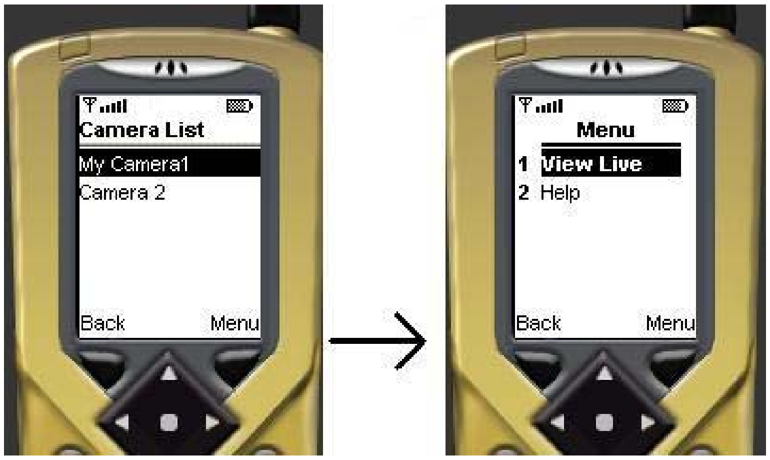


Figure 4.1-5: Screenshots - Emulator A

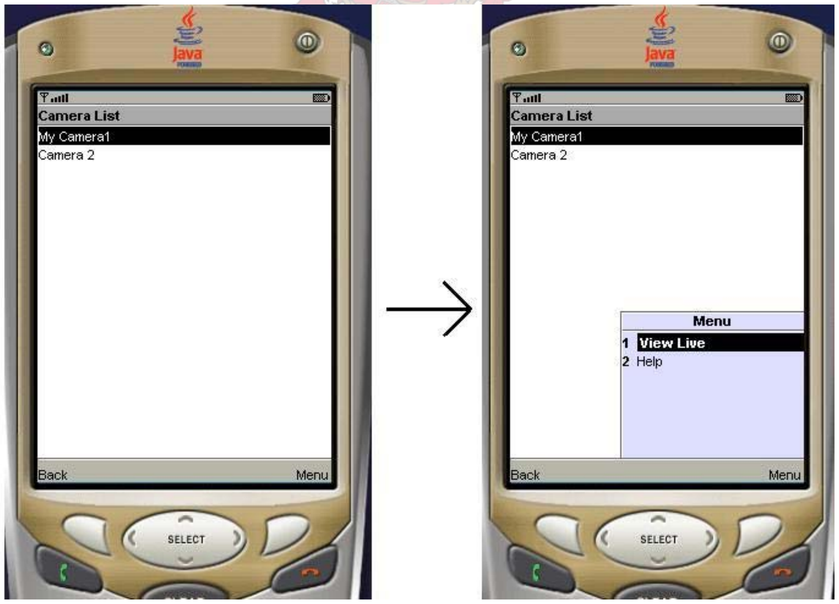


Figure 4.1-6: Screenshots - Emulator B

The user interface is therefore designed just to provide the user with the basic options, for example to exit the application, change settings and to view or stop a video stream.

4.1.3 TCP MIDlet

The TCP MIDlet is implemented in similar fashion to the HTTP MIDlet. Again, the way that TCP is implemented on the phones, must be understood before implementing it in the application.

As already discussed, TCP sockets are not supported by MIDP 1.0, but optionally supported in MIDP 2.0. A MIDP 2.0 application would typically use the SocketConnection class of the javax.microedition.io library, to implement and use sockets. In MIDP 1.0, this class is not available, however. Therefore, if SocketConnection is used, no MIDP 1.0 phones, including phones that do support sockets, can use the application.

Another approach is therefore taken, to ensure compatibility with as wide a range of phones as possible. The J2ME Generic Connection Framework (GCF) defines the connection interfaces in an inheritance hierarchy, as shown in Figure 4.1-7 [23] All connection interfaces in the hierarchy can be created, by using the open method, of the Connector class.

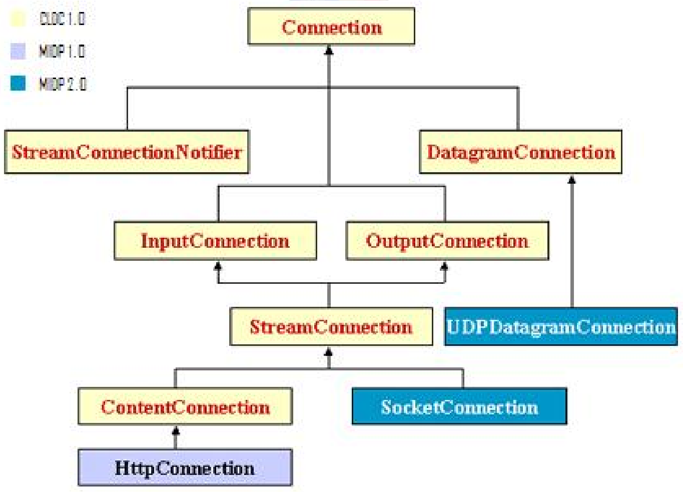


Figure 4.1-7: GCF Hierarchy

The StreamConnection interface is available in both MIDP 1.0 and MIDP 2.0 and can be used with the Connector.open class, to make a socket connection; if the underlying support in the phone's Java implementation is available. Since StreamConnection is lower in the hierarchy than SocketConnection, not all the methods and features of the SocketConnection are available and are, therefore, implemented manually. The feasibility of this approach is easily tested by implementing it on a MIDP 1.0 phone, in this case a Nokia 6820. Although the fact that it works on the test MIDP 1.0 phone model does not necessarily ensure that it will work on all MIDP 1.0 phones, it does prove the concept.

Since a persistent connection is now available in the TCP MIDlet, the approach of using sessions, as in the HTTP case, is not required. Once the TCP connection is made with the server, requests, messages and image data can be sent in both directions over the connection, without having to close the connection before the user logs off. This simplifies the MIDlet implementation significantly, compared to the HTTP version.

4.1.3.1 TCP Message Sequence Chart

The message sequence chart (Figure 4.1-8) illustrates the use of the persistent TCP connection.

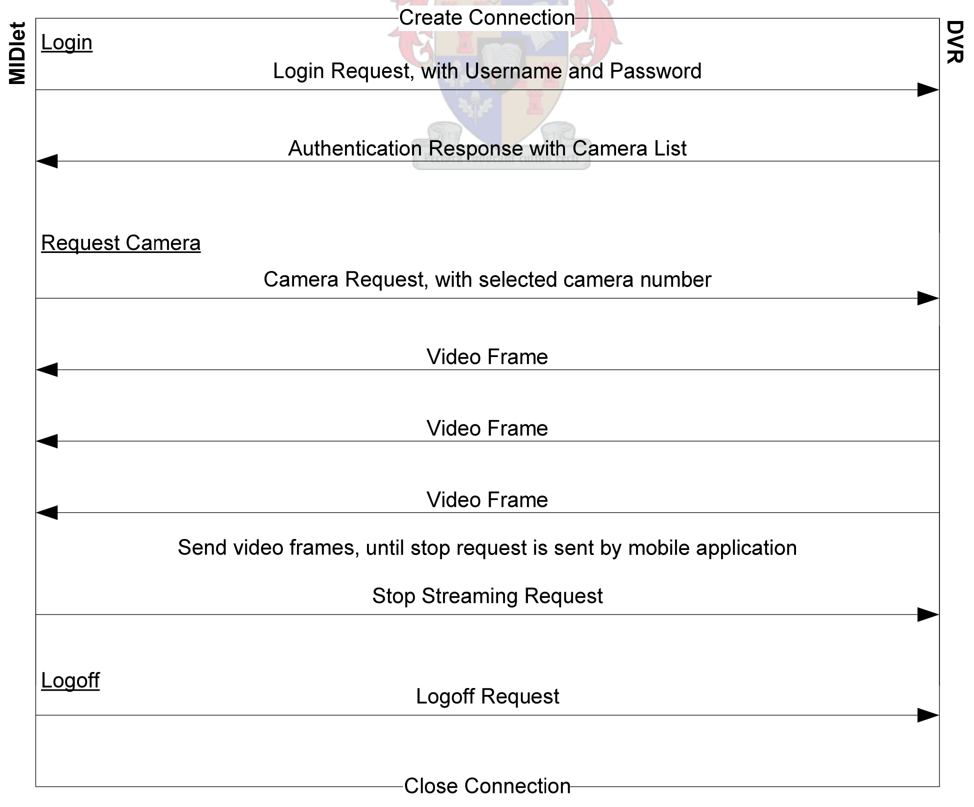

MIDlet
DVR
Create Connection
Login
Login Request, with Username and Password
Authentication Response with Camera List
Request Camera
Camera Request, with selected camera number
Video Frame
Video Frame
Video Frame
Send video frames, until stop request is sent by mobile application
Stop Streaming Request
Logoff
Logoff Request
Close Connection

Figure 4.1-8: TCP Message Sequence Chart

4.1.3.2 Implementing the TCP Communication Thread

Like the HTTP communication thread, the TCP communication thread is implemented in a separate class, which uses the CommunicationThreadInterface (Table 4.1-1) to communicate with the main thread. The basic operation principle used with the HTTP thread is also followed, but with minor adjustments, to take advantage of the persistent connection.

Authentication

When the communication thread receives the login request message, interface and DVR address from the main thread, the thread wakes from idle mode and creates the socket connection with the DVR. The login details are then sent to the DVR, the result read from the socket connection and forwarded via the interface to the main thread. The connection is not closed then, as is the case with the HTTP communication thread, but kept open when the thread returns to idle mode.

The flow diagram for the login process is shown in Figure 4.1-9

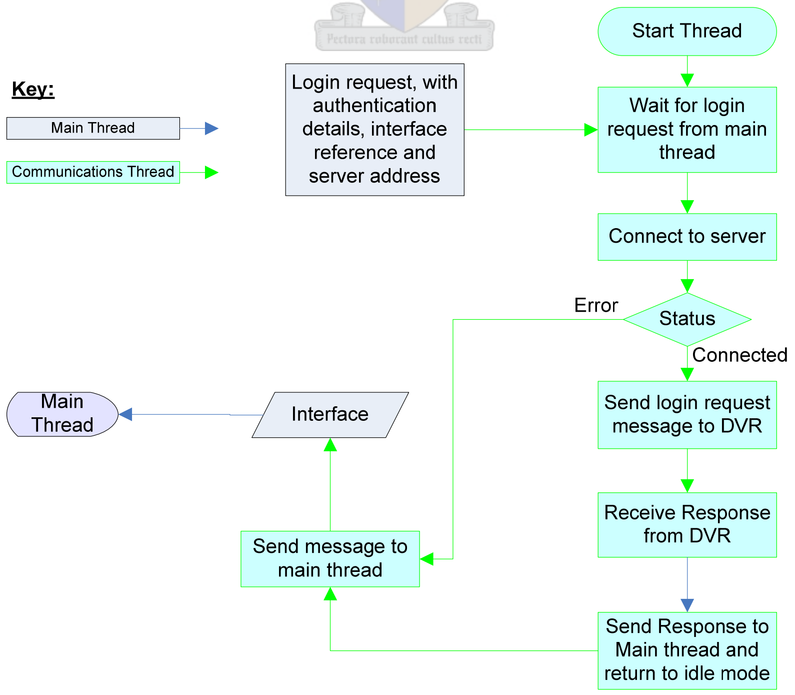

Key:
Main Thread
Communications Thread
Login request, with authentication details, interface reference and server address
Start Thread
Wait for login request from main thread
Connect to server
Status
Error
Connected
Send login request message to DVR
Receive Response from DVR
Send Response to Main thread and return to idle mode
Send message to main thread
Interface
Main Thread

Figure 4.1-9: TCP Login Flow Diagram

Video Streaming

The video streaming implementation for the TCP MIDlet is again very similar to the streaming of the HTTP MIDlet. The connection is already open when the request is sent. The connection thread sends a stop streaming request to the DVR, when no more frames are required, instead of closing the connection, as illustrated in Figure 4.1-10

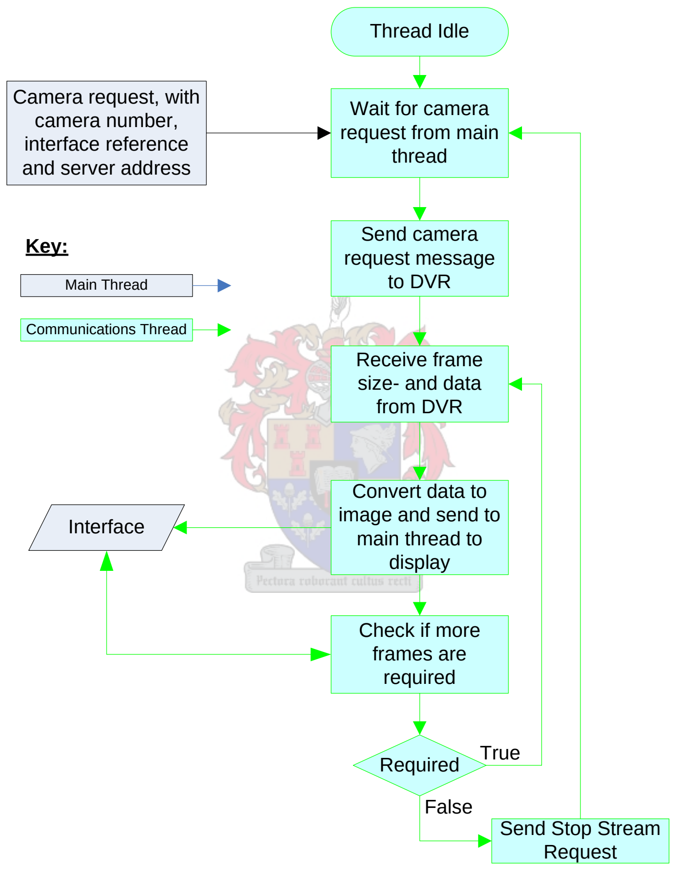

Thread Idle
Camera request, with camera number, interface reference and server address
Wait for camera request from main thread
Key:
Main Thread
Communications Thread
Send camera request message to DVR
Receive frame size- and data from DVR
Interface
Convert data to image and send to main thread to display
Check if more frames are required
Required
True
False
Send Stop Stream Request

Figure 4.1-10: TCP Streaming Flow Diagram

User Interface

The user interface for the TCP MIDlet is the same as used and described in Section 4.1.2.6, for the HTTP MIDlet.

4.1.4 MIDlet Screenshots and Testing

The initial testing and evaluation of the MIDlets are done by using the Sun Microsystems Wireless Toolkit's DefaultColourPhone cellphone emulator, in conjunction with the DVR. In the initial tests, both the HTTP and TCP MIDlets give the same results. Since both are tested on the local machine, the connection times between requests, applicable to the HTTP, are negligible. The following screenshots show the user interface and a snapshot of the video streaming on the emulator. Since the screenshots for the HTTP- and TCP MIDlets look exactly alike, only one set is shown.

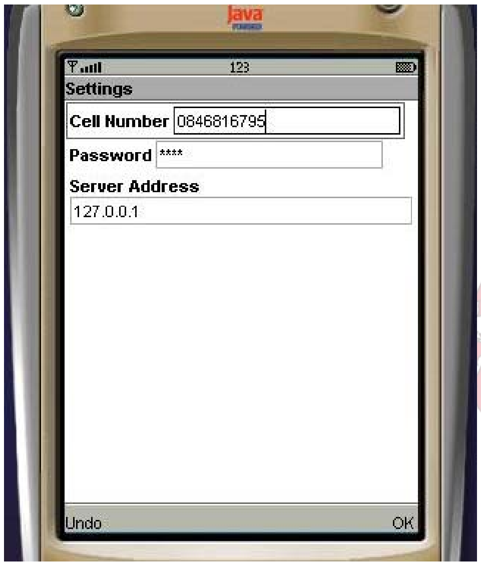

Settings
Cell Number 0846816795
Password ****
Server Address
127.0.0.1

Figure 4.1-11: Settings

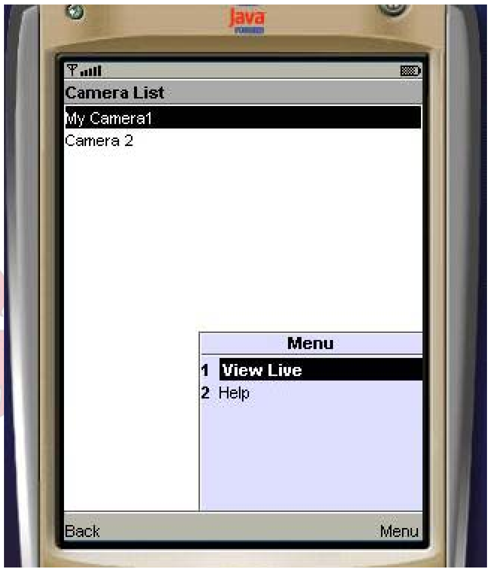

Camera List
My Camera1
Camera 2

Figure 4.1-12: Select Camera

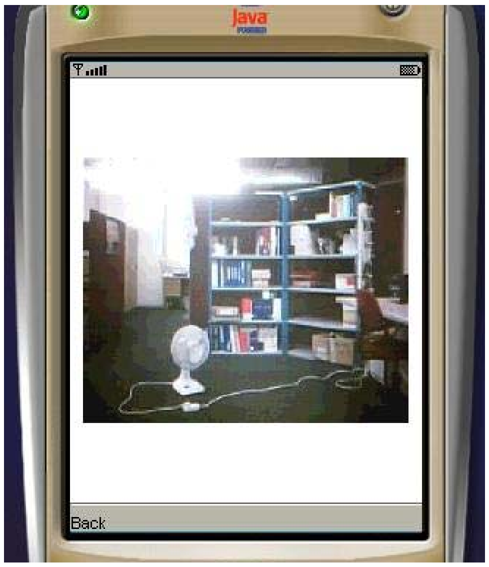


Figure 4.1-13: Video Streaming

The further testing, evaluation and comparison of the MIDlets are discussed in more detail in Chapter 5.

4.2 DVR Implementation

This section describes the implementation of the DVR. The theoretical design, discussed in Chapter 3 is the basis of the DVR program flow. The underlying design of the application is, however, described in more detail in this section. The section starts off by giving a general overview of the DVR user interface and components and then proceeds to a more detailed discussion around the streaming video implementation. Since the focus is on the implementation of the video streaming, not all elements of the DVR implementation are discussed in detail in this report and only the basic implementation principles are discussed.

4.2.1 Graphical User Interface (GUI)

The Delphi IDE, used for the implementation of the DVR, encourages the developer to start the implementation of the application, by starting with the layout and design of the GUI and to then proceed by implementing the underlying functions and technologies of the application. The main DVR GUI consists of the main application window, with a typical Windows application menu, a shortcut toolbar, message window and an area designated for the display of the cameras.

This is easily realized by using the Multiple Document Interface (MDI) application format, provided by the IDE. An MDI application consists of a main (Parent) window and an undefined number of sub-windows (Children), inside the main window. In the DVR implementation, each MDI Child is a separate window inside the main GUI, used to display and control a specific camera. Each Child window can be controlled either from the Parent window, or individually. The MDI application has built-in functions to size, show, hide, arrange or control the windows inside the main application.

This implies that, for every camera set up on the DVR, a separate Child (camera window) is created inside the MDI application. Each camera window can now have its own control interface, devices settings- and preferences as well as the camera component, running in its own thread, as described earlier.

The message window is implemented in a separate panel in the main window. The panel is provided with tabs, to switch between camera shortcuts, camera control, or message- and status view. A button is added to the shortcut toolbar, to show or hide the panel.

4.2.2 Implementing the Camera Component- and Window

The camera window consists of a camera display area and a panel with buttons to control the camera functions- and settings. The camera display is an integral part of the camera thread component described earlier, in order to display video frames in the window, without affecting the functioning of the main application, or other camera windows.

In order to minimize the amount of low-level programming required to interface with camera input devices connected to the DVR, a third-party Visual Component Library (VCL) component is used. The component selected in this case, is the TVideoGrabber object. This component gives the developer access to the camera device drivers installed on the machine. It also gives the ability to start- or stop a camera, use installed codecs to record video, access driver settings and display video on the screen. Each component runs in a separate thread, which means that the threading issue is automatically solved by using this component. The component has built-in video motion detection with the ability to notify the application when motion is detected. Individual video frames can be accessed by using the component. By using this component, the task of implementing the camera component is reduced to implementing the camera window GUI, a camera settings GUI and to use the functions of the component to detect motion and extract video frames to implement video streaming to the cellphone.

Camera Settings

When a user adds a camera to the DVR, the camera settings must first be selected and configured. The application uses the TVideoGrabber component to get a list of available video devices, from which the user can select a device. Then the application retrieves the device-specific settings (framerate, resolution, contrast, etc.) for the selected camera and displays the settings interface. A list of available recording settings is presented to the user, in which the recording framerate, compression ratio and related settings can be adjusted. Once these settings are done, the application creates a camera window, which includes a TVideoGrabber component and a record of all settings. The record is saved on the hard drive, to enable the application to automatically load the camera settings when the application is restarted.

Figure 4.2-1 illustrates how the component is used in the application.

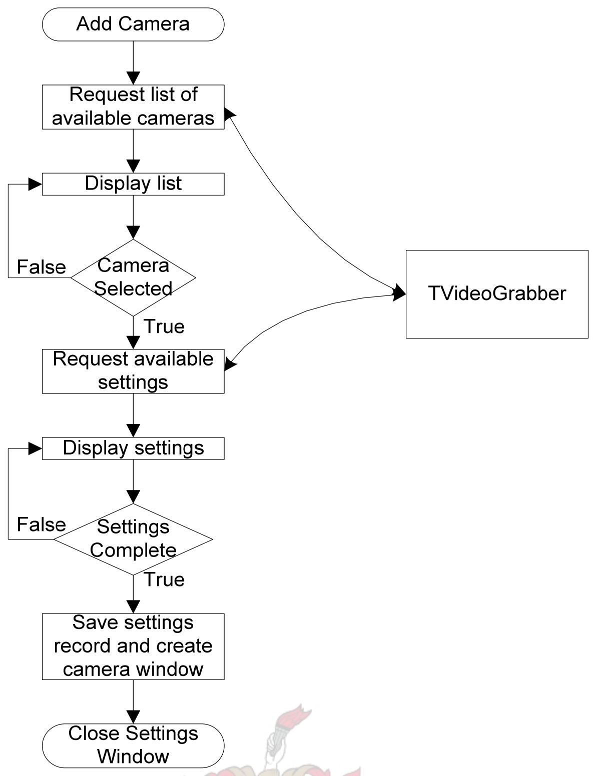


Figure 4.2-1: Camera Settings Flow Diagram

Motion Detection

The user is given the option to activate- or disable motion detection, or to set a motion detection interval. A timer is used to implement the motion detection interval of each respective camera. Once motion is detected, the timer is activated by the application and any subsequent motion detections ignored, until the timer is reset at the end of the preset time. The user is also given the option to only record video when motion is detected.

The TVideoGrabber component allows different sensitivity levels to be set for different areas in the camera screen. The user is presented with a snapshot of the camera view. A grid is placed on the screen, with the sensitivity level indicated for each block in the grid.

When the camera is started and the TVideoGrabber component detects motion, an OnMotionDetected event is triggered. The application then extracts video frames or activate video recording by using the TVideoGrabber component. The individual frames can be used to send with an email notification, to an email address preset by the user. A timer is used to stop the recording after a preset time, when motion-activated recording is used. The motion detection procedure is illustrated in Figure 4.2-2

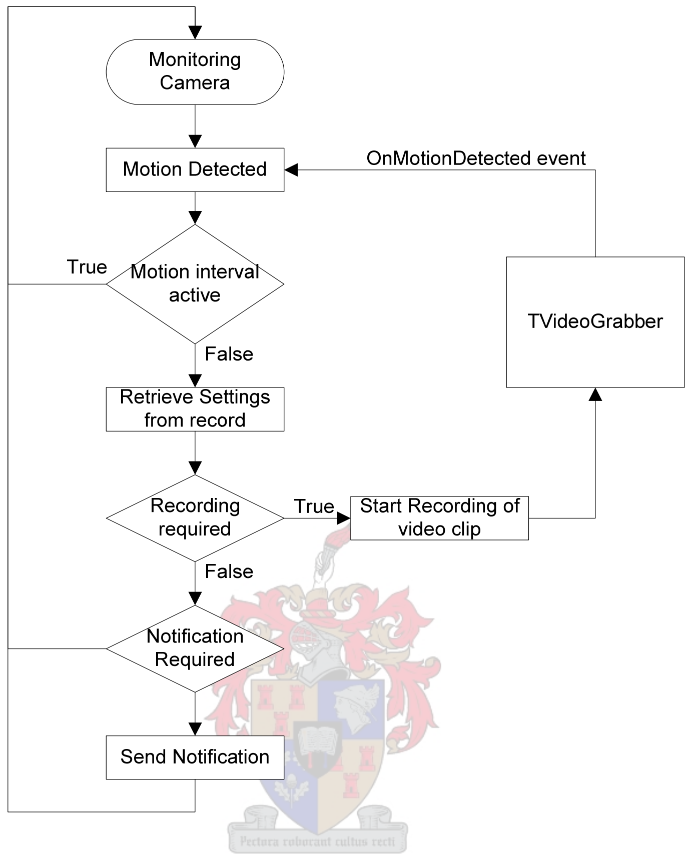


Figure 4.2-2: Motion Detection Flow Diagram

Start Monitoring

When the user starts a camera, the application retrieves the settings of the selected camera and forwards the device-, motion detection- and recording settings to the TVideoGrabber component. Before recording is started, a filename is created by the application and a directory created for the video file recorded by the camera component. If motion activated recording is set for the device, recording is not started before motion is detected, but the device started in view-only mode. If continuous recording is selected, the video recording is started, encoded with the selected video codec, by using TVideoGrabber. The monitoring program flow is shown in Figure 4.2-3

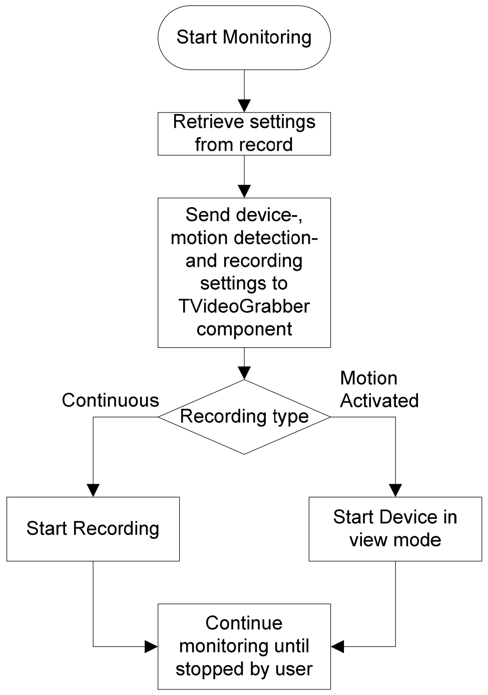

Start Monitoring
Retrieve settings from record
Send device-, motion detection- and recording settings to TVideoGrabber component
Recording type
Continuous
Motion Activated
Start Recording
Start Device in view mode
Continue monitoring until stopped by user

Figure 4.2-3: Start Monitoring Flow Diagram

4.2.3 Implementing the Communications Component and Video Streaming

The communication between the DVR and Cellphone requires the use of both sockets and HTTP, as previously discussed. Therefore, both protocols are implemented on the DVR. This section describes the implementation of the protocols and streaming video in more detail.

4.2.3.1 Implementing TCP and HTTP on the DVR

The implementation of TCP and HTTP communications in a Delphi application is simplified by using one of the various third-party components available. The open source Indy communication library, which ships with Delphi, is selected to implement the DVR communication component. The Indy components include TCP server components, HTTP server components, Email-related components, as well as helper components to make the developer's task easier.

Although an Indy HTTP server component is available, the TCP- and HTTP communication and streaming are both implemented by using a single TCP server component. This is done in order to have more control over the functioning of the HTTP implementation on the DVR, particularly to make it easier to implement the non-conventional HTTP communication required by the HTTP MIDlet. A further advantage is that only a single port needs to be opened to the Internet, instead of separate ports for the TCP and HTTP communication.

Similar to the TVideoGrabber component, the Indy TCP Server component, TIdTCPServer, also makes the task of multithreading easier. For each connection made to the application, the server component creates a separate thread (TIdPeerThread). Although the developer must still implement measures to ensure that thread interlocking does not occur, the process is simplified by the use of events, generated by the server component, to notify the main thread of any connection related actions.

Client Connect

When a client makes a connection to the DVR, an OnConnected event is triggered and a reference to the applicable TIdPeerThread given to the application. This allows the DVR to know when a client connects to the DVR and to proceed to receive further requests from the client. For any subsequent action initiated by the client, an OnExecute event is triggered, again with a reference to the applicable thread.

Once a client connects to the DVR, the DVR proceeds to read data from the connection. This data is then evaluated in order to determine if the connection is made by an HTTP MIDlet, TCP MIDlet, or another device.

The protocol defined for the login of a valid streaming client, requires a login request to be sent, as the first communication between the MIDlet and the DVR, in the case of the TCP MIDlet. An HTTP header, including a valid session ID in the HTTP header, or a header followed by a login request, is required as first communication in the case of the HTTP MIDlet. The DVR recognizes an HTTP request if the first line of data received, starts with “GET”, or “POST” and then proceeds to parse the HTTP header. If any other data is received after the connection is made, the client is not a valid streaming video MIDlet and the connection terminated by the DVR.

In order to keep control of the threads and client states, the DVR keeps a list of all clients and client states, connected to the DVR. A dynamic array of TClient records is used to implement the list. Each TClient is a record of several variables and pointers, which holds the information and streaming related data for the particular client. Since the communication threads all need access to the list of TClient's in the main thread, all access to the list is serialized, to prevent thread interlocking. The TClient record is implemented as follows in the Delphi code:

```
type
  TClient   = record                                //holds client connection data..
    PeerIP              : string[15];               { Client IP address }
    Connected           : TDateTime;                { Time of connect }
    LastAction          : TTimeStamp;               { Time of last activity }
    StreamCamNum        : Integer;                  { Index of camera viewed }
    Authenticated       : Boolean;                  { True if logged on }
    Thread              : TIdPeerThread;            { Pointer to thread }
    StartRqst           : Boolean;                  { True if video stream requested }
    StopRqst            : Boolean;                  { True if stop stream requested }
    DiscRqst            : Boolean;                  { True if client logoff requested }
    ClientName          : String;                   { Client name }
    ClientProtocol      : Integer;                  { HTTP = 02; TCP = 01 }
    SessionID           : String;                   { Session ID }
    ChunkedTransfer     : Boolean;                  { HTTP Transfer type }
end;
```

Once a valid MIDlet login request is received, a TClient record is created and the TClient fields initialized and updated as applicable. The TClient record is then added to the client array. The index of the record in the array is stored in the communication thread, to enable subsequent events from the same thread, to be able to find the correct TClient record in the array of clients. The authentication of the client follows. If the login is valid, a confirmation message with a list of available cameras is sent to the client, via the connection thread. If the client is an HTTP MIDlet, a unique session ID is created and sent in an HTTP header before the message. The HTTP client is disconnected after the message.

If an HTTP MIDlet connects and the header contains a valid session ID, it means that the client is already logged on to the DVR and a TClient record for the client is already in the list. A new TClient record is, therefore, not created. A valid session ID is located by searching for a matching session ID in the client list, which also returns the index of the client in the array. The LastAction field is updated with the new connection time when the index is found, after which the thread waits for further data to be received from the connection.

The client connection and authentication flow diagrams are shown in Figure 4.2-4 and Figure 4.2-5 respectively.

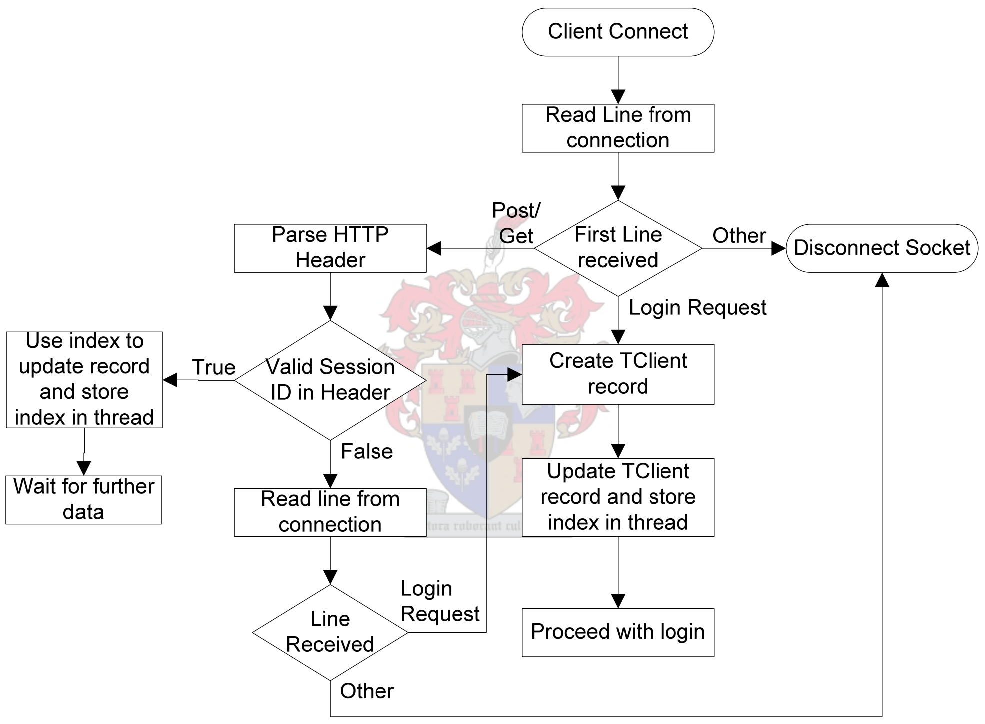


Figure 4.2-4: Client Connection Flow Diagram

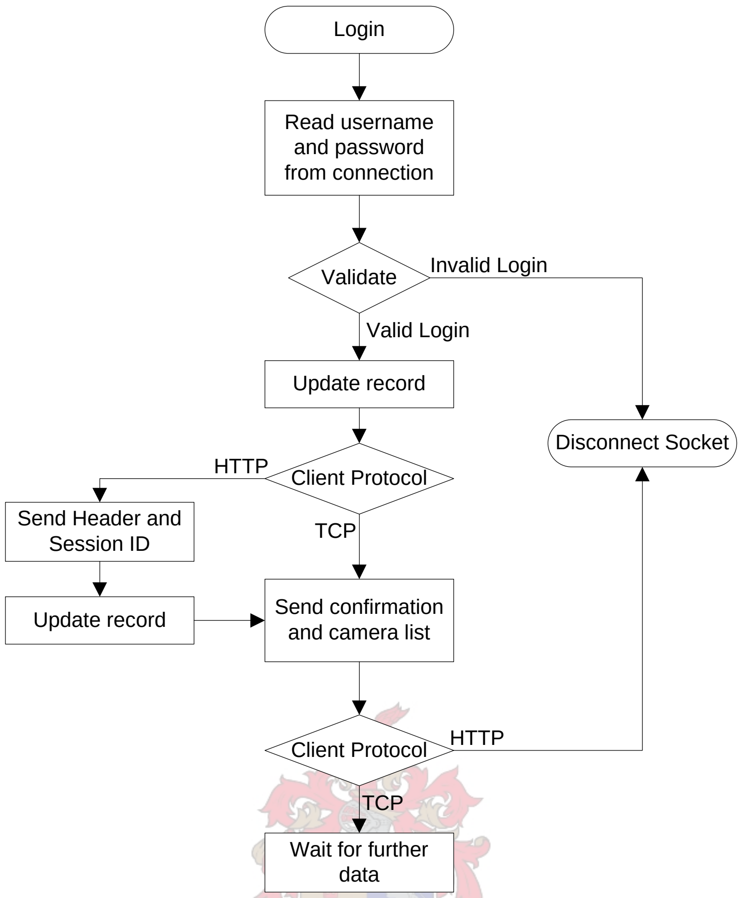


Figure 4.2-5: Client Login Flow Diagram

Client Logoff

When a TCP MIDlet logs off from the DVR, a logoff request is sent to the DVR, but a socket disconnection is also regarded as the client being logged off. An HTTP client is logged off only if the logoff request is received, or when the HTTP session expires. Once a client is logged off, its connection thread is terminated and the TClient record removed from the client array.

4.2.4 Implementing Video Streaming

Once a client is logged on to the DVR, a request for a video stream can be sent to the DVR. The stream request is received from a connected and authenticated client, when the OnExecute event is triggered. In the case of a TCP client, the stream is sent to the client directly following the request. In the case of an HTTP client, an HTTP response header precedes the video stream. The remainder of this section focuses on the activation of the camera, processing of the video frame and transmission of the frames (images) to the client, until the client requests the stream to stop (TCP client) or the socket is disconnected (HTTP client), as described in Section 4.1

Camera Request

When the OnExecute event is triggered, the TClient record index is retrieved from the Thread. The request is then read from the connection and the camera number parsed from the request message, if a camera request is received. The TClient record is then updated with the camera number and the StartRqst variable set to true, before a Thread.synchronize(CheckRequests) call is made to the main thread, to check all pending client requests. This approach is used for the start streaming-, stop streaming-, and logoff requests, because of the fact that no parameters can be passed with the synchronize method. When the synchronized procedure is called, the DVR loops through the TClient records, executes the requests and resets the request flags. When a camera request is received, the DVR retrieves the camera number of the requesting client record and sends a request to start streaming, to the camera window.

The camera window keeps count of the amount of stream start- and stream stop requests that is received. The camera starts, with the TVideoGrabber's frame grabber enabled whenever the number of clients is greater than zero and stops, whenever the number of clients equals zero, unless recording is active.

The flow diagram for starting the video streaming, is shown in Figure 4.2-6

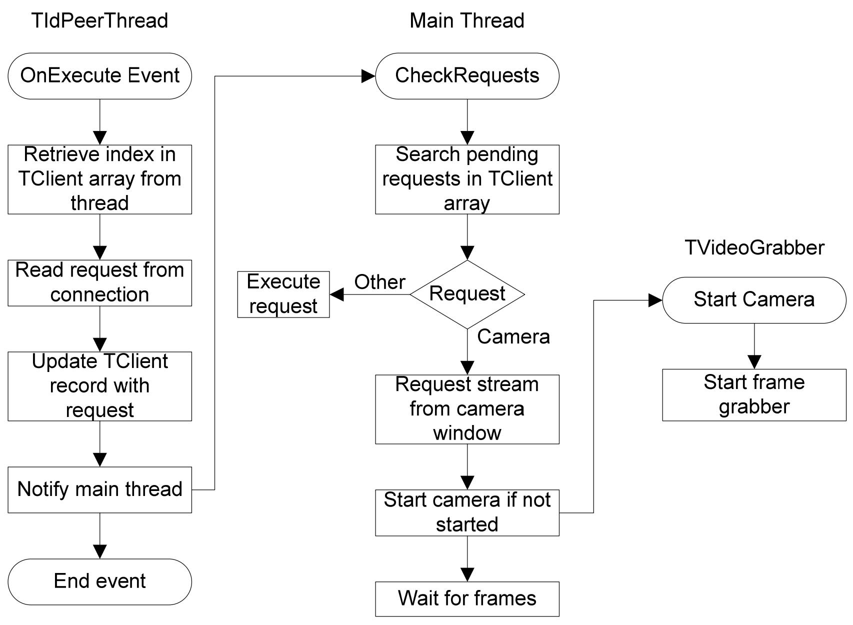

TIdPeerThread
Main Thread
TVideoGrabber
OnExecute Event
Retrieve index in TClient array from thread
Read request from connection
Update TClient record with request
Notify main thread
End event
CheckRequests
Search pending requests in TClient array
Request
Other
Execute request
Camera
Request stream from camera window
Start camera if not started
Wait for frames
Start Camera
Start frame grabber

Figure 4.2-6: Start Video Streaming Flow Diagram

The TVideoGrabber triggers an OnFrameCaptureCompleted event for each video image captured to memory and passes the reference to the application. The application then processes and converts the image into a suitable format (JPEG or PNG, as set by the user), for sending via the client connection thread. In the conversion process, the image is sized to either a resolution preset by the user, or to make the image data size a certain preset size. The user can select which option to use. The application first sends the size of the image (integer, followed by carriage return character) and then the image data, to the client. The DVR does not send a subsequent image to the client, before the Thread.Connection.Writebuffer procedure, which is used to send the image data, is completed for the last image sent. The image data is sent to each client that has the relevant camera selected for streaming.

The video streaming flow diagram is illustrated in Figure 4.2-7

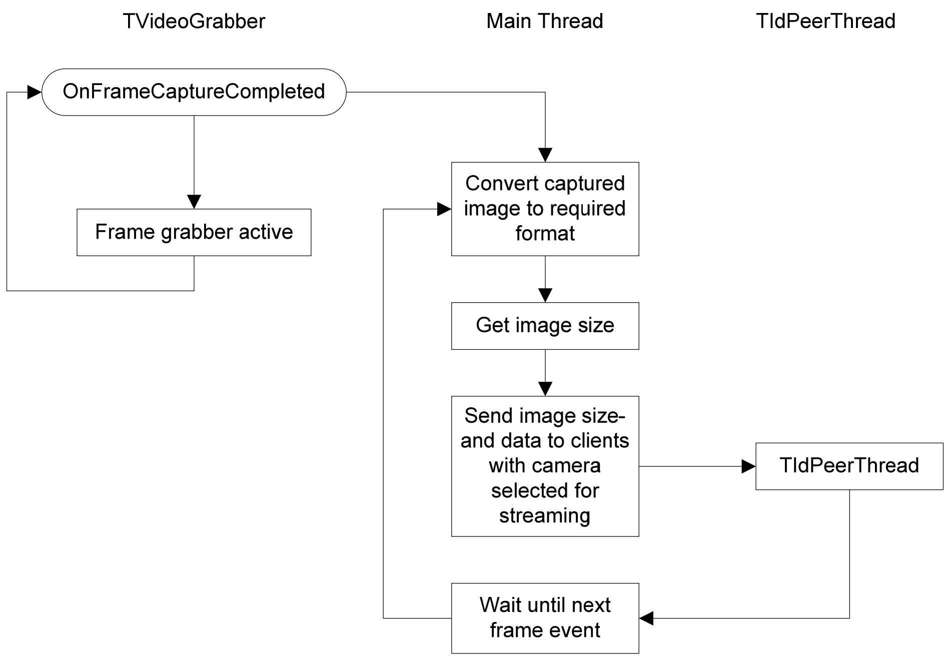

TVideoGrabber
Main Thread
TIdPeerThread
OnFrameCaptureCompleted
Frame grabber active
Convert captured image to required format
Get image size
Send image size- and data to clients with camera selected for streaming
TIdPeerThread
Wait until next frame event

Figure 4.2-7: Video Streaming Flow Diagram

Managing the Client List and Connections

In testing the application over the cellular networks, in some instances the connection thread does not trigger the OnDisconnect event when the TCP version cellphone application is exited. Although this only happens on rare occasions, it results in TClient records being kept in the client list, even when the actual client is not connected to the DVR anymore. The cause of this is unknown and suspected to be related to the cellular networks. Therefore, a 'cleanup system' is added to the DVR, to terminate threads that are inactive for a certain time.

The cleanup is activated by using a timer, to check all threads in the client list, terminate inactive threads and to remove the applicable TClient records from the array. This system is also used to remove HTTP clients from the list, when the session timeout, described earlier, is reached. The cleanup procedure is executed every 10 seconds, by the timer.

4.2.5 Implementing the Review Window

The video review unit is implemented in a separate form, by using a TVideoGrabber component in the form, for playback. The GUI provides the user with the ability to select the recorded video to be reviewed. Since the installed Windows codecs are used, no export functionality is required for the review unit.

4.2.6 Email Alarm Notifications

When motion is detected and the user has selected that email notifications must be sent when an alarm condition occurs, the Indy SMTP client component is used to compile and send an email notification to the preset email address. A video clip, or video images, captured by using the TVideoGrabber component, in combination with a timer, can be attached to the email.

4.2.7 Additional DVR Functions

Various additional elements of the DVR are implemented in order to make the application more user friendly, to provide accurate logging of all client connections, DVR actions and alarms and to add mobile users with usernames and passwords. These functions are implemented using standard programming- and database techniques. Since the focus of this Thesis is oriented more towards the implementation of streaming in a mobile environment, the details of these elements of the DVR are not discussed in further detail in this document. The reader is referred to *www.torry.net*, for further information about the Delphi language, the implementation of databases, GUI design and the use of threads in the applications, as related to this project.

4.3 <u>Implementation Summary</u>

The implementation of the phone- and DVR applications were discussed in this chapter. The program flow diagrams and descriptions of the major components of the software were presented. It was described how two versions of the cellphone application have been implemented for, respectively, TCP and HTTP compatible phones. The DVR is implemented by using the Delphi IDE and utilizing several third party components.

The evaluation of the phone- and DVR applications, measured against the thesis objectives, is discussed in the following chapter.

5 Evaluation and Results

In this chapter, the designs and implementations derived in the previous chapters, are evaluated against the thesis objectives, presented in Section 1.1. The chapter starts off by evaluating the phone- applications, using emulators. The applications are then tested on real phones, over a real cellphone network. Screenshots of the emulated results and photos of the cellphone applications on real phones are provided.
The DVR application is then evaluated and screenshots provided. The email alarm notification mechanism is also demonstrated.

5.1 <u>MIDlet Evaluation</u>

5.1.1 Simulated Results

The Sun Microsystems cellphone emulator is used to test the applications over a mobile broadband network. A Nokia cellphone is used to connect the PC, on which the emulator is run, to the Internet, using GPRS- and 3G connections. The DVR application is installed on a PC connected to the Internet, via a 512 kbps ADSL connection. A framerate (fps) counter, developed by Wayne Forrest [32], is included in the MIDlet and used to calculate and display the streaming video framerate on the screen of the MIDlet.

Table 5.1-1 indicates the average times to connect to the DVR, measured from the moment the connect button is pressed, until the connection is made with the DVR. The table also indicates the average times measured, between selecting a command and the command being received by the DVR. The maximum- and average framerates achieved are also indicated. The values represent the average figures from five consecutive tests, for each scenario.

Table 5.1-1: Emulation Results

Description	Average
HTTP MIDlet over 3G	
Connection Time	1-3 s
Request Time	1 s
Maximum Framerate	9 fps
Average Framerate	7 fps
TCP MIDlet over 3G	
Connection Time	1-2 s
Request Time	<1 s
Maximum Framerate	8 fps
Average Framerate	7 fps
HTTP MIDlet over GPRS	
Connection Time	4-7s
Request Time	3-5s
Maximum Framerate	3 fps
Average Framerate	1 fps
TCP MIDlet over GPRS	
Connection Time	3-4s
Request Time	2-3s
Maximum Framerate	3 fps
Average Framerate	1 fps

Although the results in Table 5.1-1 are useful to demonstrate the difference in performance between the different scenarios, it cannot be used as an accurate measurement of the expected framerates and latencies. Due to the fact that the underlying network characteristics, of both the cellular- and ADSL networks are unknown, various variables are excluded from the result. Both ADSL and the cellular networks are contended services and influenced by the amount of users using the network at a given time, distance from the base station and signal strength of the cellphone network. These figures also tend to fluctuate in different areas, at different times of the day and week. Therefore, in order to compile more accurate statistics, multiple tests are required, at different times of the day, different days of the week and at different locations. This is, however, an investigation beyond the scope of this project and therefore, not included in this Thesis.

Due to the way the streaming is protocol implemented, the image quality of all frames sent from the DVR is constant, irrespective of the framerate and network throughput. The visual quality of the video displayed on the emulator is, therefore, the same for the TCP- and HTTP MIDlet's, as well as for using the GPRS- and 3G network connections. The screenshot for one of the four emulations is, therefore, the same as for the others. An example of the video streaming emulation is shown in the following screenshot (Figure 5.1-1).

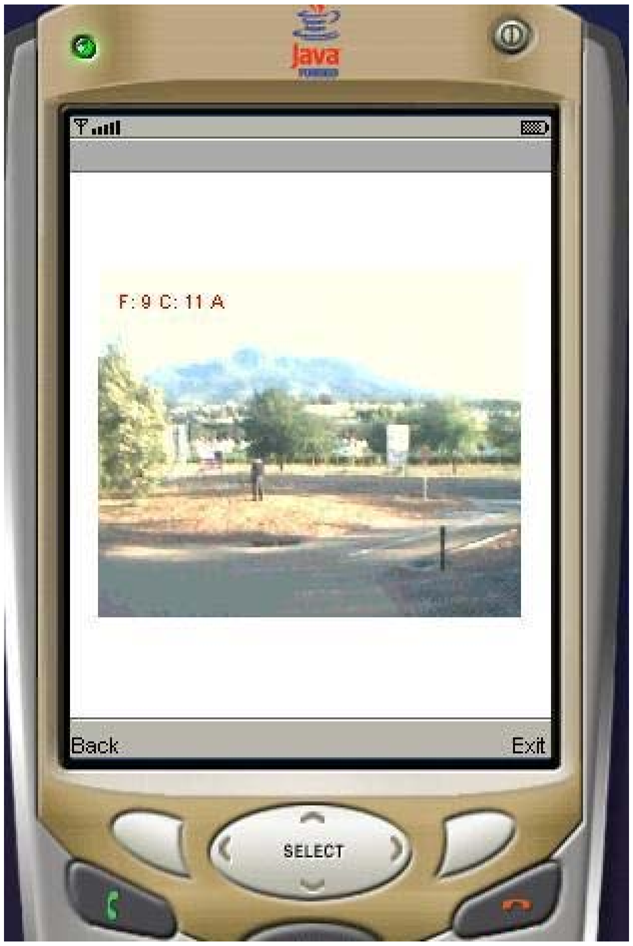


Figure 5.1-1: MIDlet Video Streaming Screenshot

5.1.2 Results on Real-life Cellphones

The HTTP- and TCP MIDlet's have been tested on MIDP 1.0 and MIDP 2.0 cellphones. The phones used, are a Nokia 6820 (MIDP 1.0) and a Nokia 6630 (MIDP 2.0). The framerates and latencies measured, using the real phones are not tabled in detail, due to the numerous unknowns in the network. The connection times of different phones differ, as well as the network latencies of the cellphone networks, at a given time of day and at different distances from the base stations. Furthermore, both the back-end connection (ADSL) and the cellular phone network are subject to contention ratios, which are totally unknown to the user. The real life tests are therefore limited to the perceived video quality- and usability on the phones.

TCP MIDlet

The TCP MIDlet works on both cellphones, as expected. The connection time is significantly longer, for both the 3G enabled Nokia 6630 and the EDGE/GPRS enabled Nokia 6820, than perceived for the emulator. This can be attributed to either the network conditions, at the time of testing, or the implementation of the connectivity in the Java runtime environment on the phone. The time for camera requests is, however, similar to the emulation results.

HTTP MIDlet

The HTTP MIDlet works on the Nokia 6630, as expected. However, on the MIDP 1.0 Nokia, it does not. A connection can be made to the DVR and a camera list received, but no video stream is received when the camera is selected, even though the DVR indicates that it is sending images to the cellphone. This is contrary to the expected result, which is that at least the HTTP version should be able to work on MIDP 1.0- and MIDP 2.0 phones, while the TCP version should work on most MIDP 2.0- and some MIDP 1.0 phones.

Further research reveals that various developers have encountered this problem, with some of the Nokia phone models, where an HTTP response is only received by the phone, once the response is fully sent by the DVR and the connection closed. This can be attributed either to a proxy being present between the application and the DVR, or the way that Nokia implements the HTTP protocol on some of the handsets. Further research and testing of different handsets is required in order to get a clearer understanding of this behavior. It is however relatively safe to assume that, since the TCP MIDlet works on the MIDP 1.0 Nokia phone, it should work on most of the newer Nokia phones as well. Whether this holds true for other phone manufacturers as well, is uncertain and can again only be established by testing the application on a variety of phones. This is, however, a significant practical undertaking that will not really contribute to the fundamental design principles of the project as is, as such, beyond the scope of this thesis.

Visual Results

Photos of the streaming on the MIDP 1.0 and 2.0 cellphones are shown in Figure 5.1-2 and Figure 5.1-3. Although the quality of the photos do not do justice to the actual quality perceived on the handsets, it gives an idea of the streaming appearance on the phones.

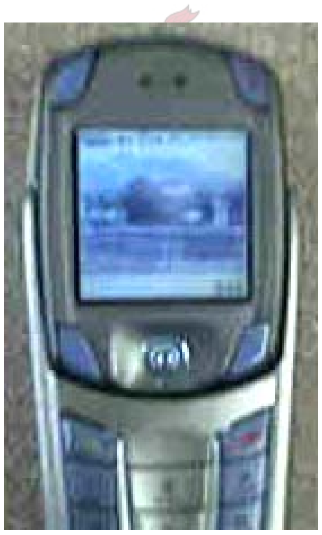


Figure 5.1-2: Video Streaming on MIDP 1.0 Cellphone

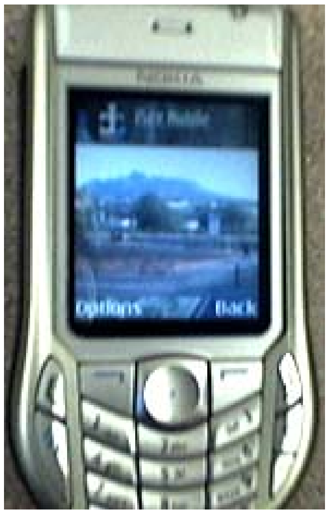


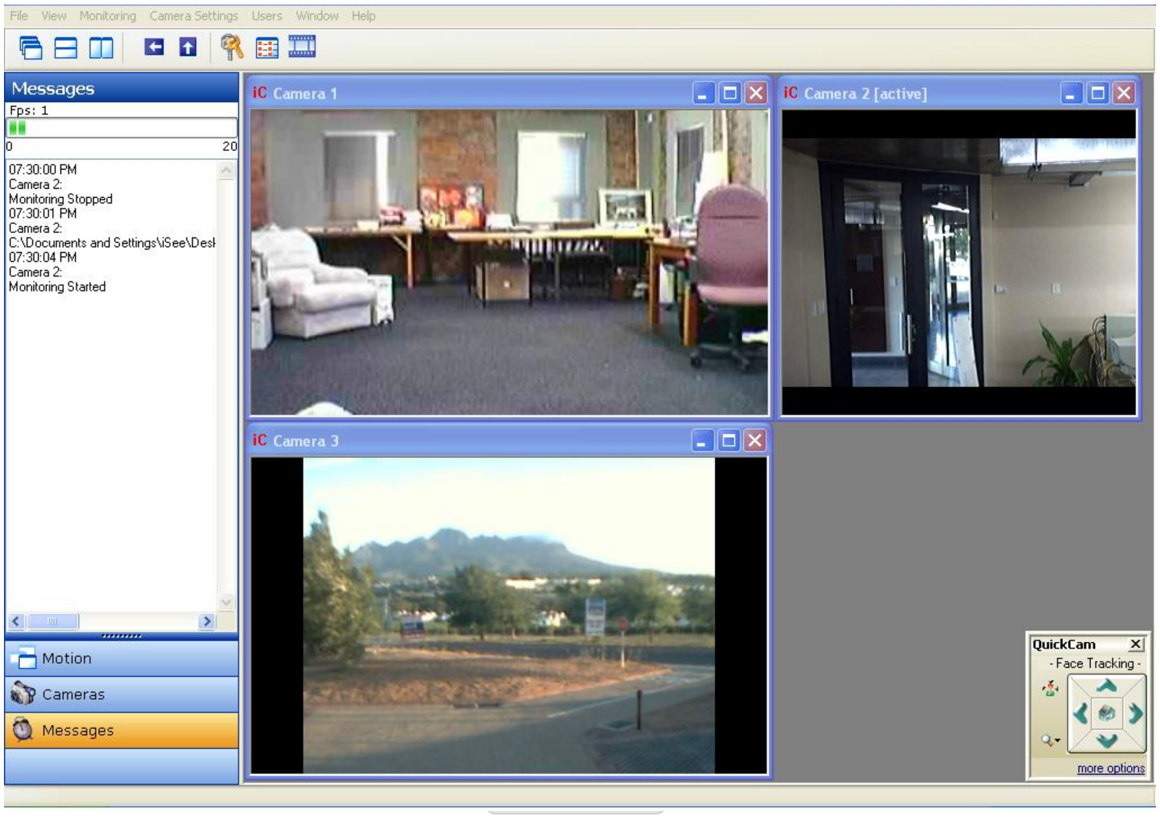
Figure 5.1-3: Video Streaming on MIDP 2.0 Cellphone

5.2 DVR Evaluation

The evaluation of the DVR is relatively simple, if compared to the Thesis goals, but also complex, because of the fact that a user friendly application is perceived differently by different users. The basic functions required by the Thesis objectives are included in the implementation, and streaming to both MIDP 1.0 and MIDP 2.0 phones is successfully demonstrated. The vast variety of phones on the market makes it extremely difficult, however, to know for certain if the application will work on most phones, or not. The basic principles and careful implementation of the video streaming protocol as devised should, however, ensure that most phones are compatible. The newer the phone, the more likely it is that the Java implementation on the phone supports TCP and thus will be able to use the TCP MIDlet. Since the MIDlet is implemented, using TCP sockets in the most basic way possible, it should theoretically be able to work on most phones on which sockets can be used. This can only be proven by testing the application on all available phone models, which is clearly an impossible task.

Visual Results

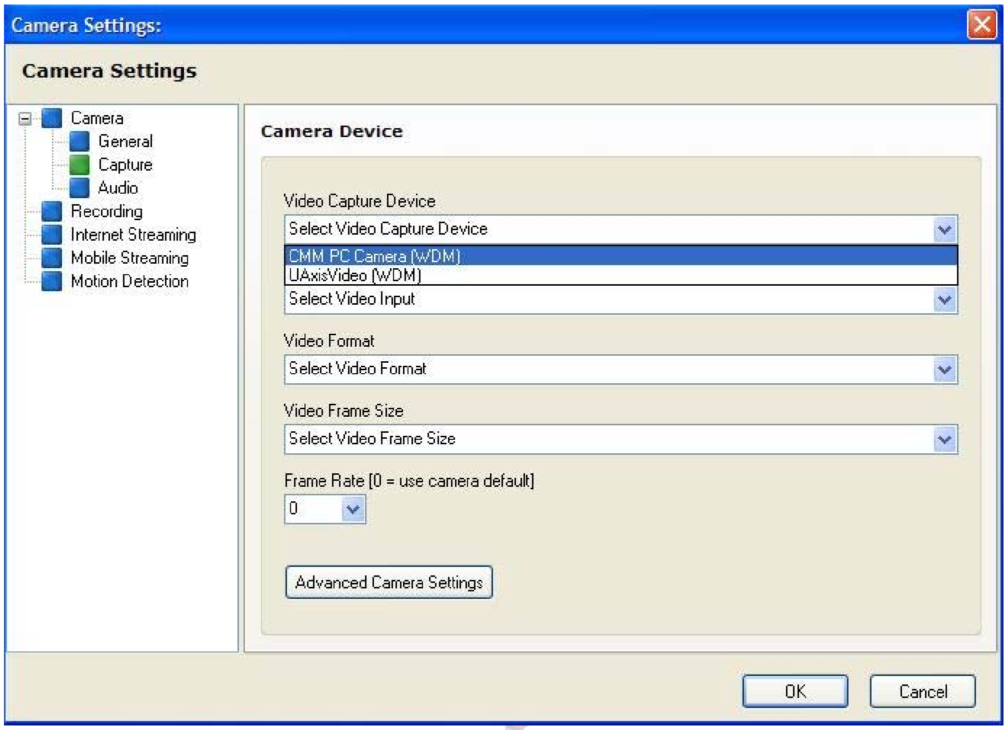
The following screenshot shows the GUI of the DVR, when monitoring three cameras (Figure 5.2-1).



```
07:30:00 PM
Camera 2:
Monitoring Stopped
07:30:01 PM
Camera 2:
C:\Documents and Settings\iSee\Desk
07:30:04 PM
Camera 2:
Monitoring Started
```

Figure 5.2-1: DVR Monitoring

The following screenshot shows the settings GUI, when selecting a camera device (Figure 5.2-2).



Video Capture Device: Select Video Capture Device
CMM PC Camera (WDM)
UAxisVideo (WDM)
Select Video Input
Video Format: Select Video Format
Video Frame Size: Select Video Frame Size
Frame Rate [0 = use camera default]: 0

Figure 5.2-2: DVR Settings Window

The alarm notification email, sent by the DVR, contains 5 images of the motion detected, each taken 1 second apart. The email body contains general information about the alarm and the camera that detected the alarm. The following shows the email contents and photos of an alarm notification, as received on the cellphone:

From: "DVR" <email@email.co.za>
Subject: Alarm!
To: email@email.co.za
Motion was detected by camera Camera 1, Office Camera
at 07:36:54 PM on 2007/01/08
Please see attached pictures of detected motion

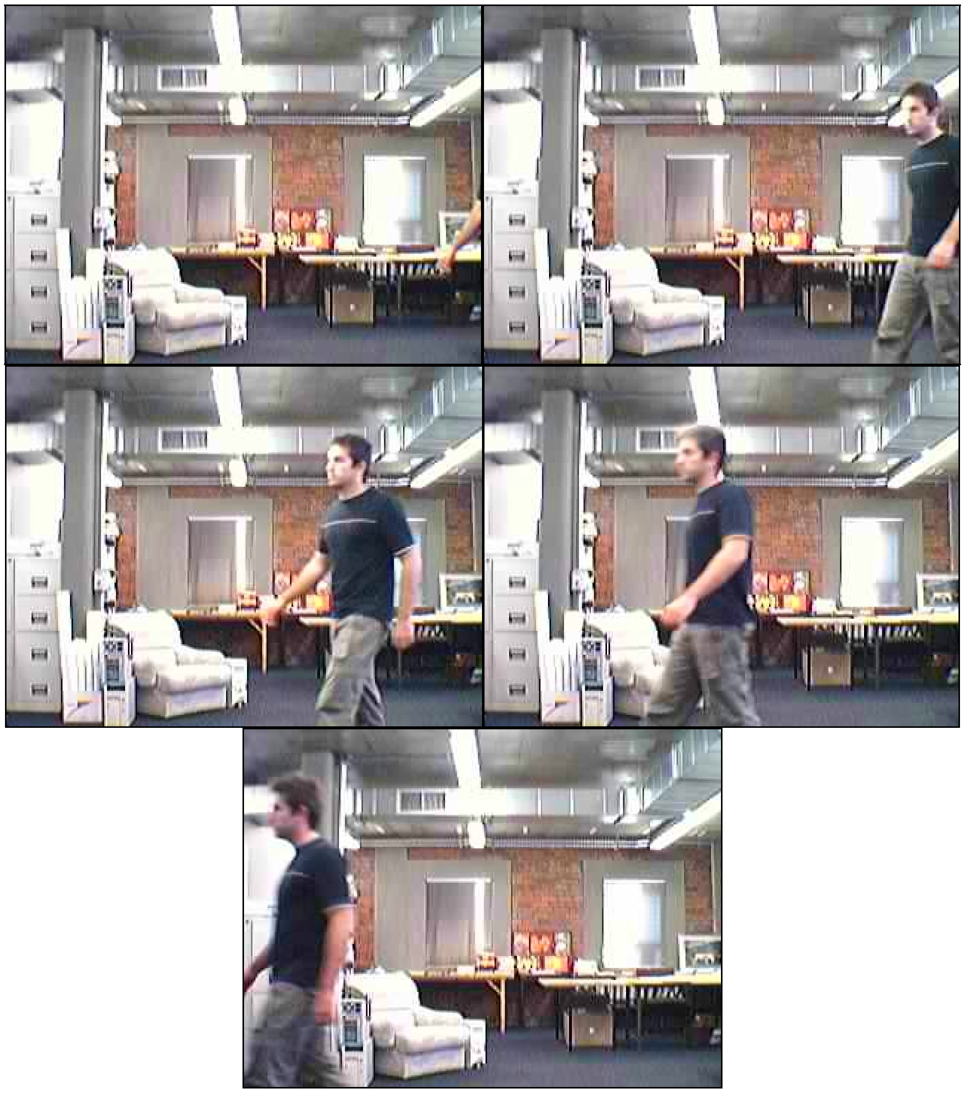


Figure 5.2-3: Motion Alarm Images

5.3 <u>Evaluation Summary</u>

The implementation of the DVR and MIDlet can be considered successful, since full, stable functionality is proved. As a result, the thesis objectives are also met. Although the HTTP MIDlet does not work on the Nokia 6820, streaming is achieved to the phone by using the TCP MIDlet. Therefore, streaming to both MIDP 1.0 and MIDP 2.0 phones, is achieved and since these are currently the only specifications of J2ME, it is safe to assume that the applications should work on all phones, if all phone manufacturers implement the MIDP 1.0 and MIDP 2.0 specifications correctly. Since this is not necessarily the case due to undocumented deviations by manufacturers, absolute certainty can only be obtained by testing all phones. A possible solution, for phones which do not support the HTTP- or TCP streaming implementations, could be to let the phone application download small sequences of video clips or images, by using a repeated sequence of requests to the DVR. If such an implementation could still be regarded as live video streaming is, however, debatable.

In summary, it is clear that the evaluation seems to satisfy the initial design approach taken.

6 Contributions, Conclusions and Future Work

The rapid way in which the mobile phone environment is advancing and the improving capabilities of cellphones and mobile networks, combined with the increased need for remotely accessible video surveillance systems, has presented the realistic possibility of utilizing cellphones as an additional monitoring element. To this end, this project has made the following contributions:

6.1 Contributions

- In this Thesis, the various characteristics, capabilities and limitations of mobile networks- and devices, as well as the various options for streaming video content to cellphones were investigated in depth. This knowledge was used to present the design- and implementation of a digital video surveillance system, with the ability to stream live video to cellphones.
- A finely balanced design approach was followed to attempt maximum compatibility between hardware and network capabilities and to reconcile sometimes contradictory requirements.
- A constant image quality, variable framerate video streaming protocol was successfully developed and demonstrated.
- A comprehensive DVR was also developed and implemented to host the on-site surveillance software.
- A third party movement detection algorithm was adapted and for this specific implementation and integrated into the DVR software suite.
- A client database was implemented on the DVR to provide full practical functionality.
- Full streaming functionality to both MIDP 1.0 and MIDP 2.0 enabled phones has been achieved and demonstrated.

6.2 Conclusions

Manufacturers' idiosyncrasies, marketing policies and extremely rapid development of cellphone hardware and software, presented a virtual labyrinth of protocol options, implementations, partial implementations and undocumented traps for the unwary. A careful path had to be followed through this landscape to ensure as seamless an operational system as possible. A major obstacle was to obtain exact details regarding embedded phone software. This is generally not forthcoming from manufacturers.

It was shown that although only two J2ME specifications exist, the implementation thereof, on different phones, is not guaranteed to follow the specifications 100%. The video streaming protocol is implemented in such a way, however, to try to maximize the amount of phones that will be compatible. In some instances, some minor changes to the application might be needed, which could possibly imply that there should be various different versions of the phone application for different phones. The last-resort solution, mentioned at the end of Chapter 5 is to send multiple video sequences to the phone, instead of a video stream. Although this would probably not qualify as live video streaming per definition, it could still provide the user with the same functionality.

Interesting, unsuspected and unacknowledged bugs also came to the fore from the mobile networks themselves. In spite of these, it is felt that satisfactory functionality was achieved without unacceptable compromises and by implementing workarounds where at all possible.

The project was challenging, but very interesting and its successful completion up to this point, would seem to present subsequent commercial possibilities.

6.3 Future Work

Although the DVR and cellphone applications can be used as a workable video surveillance solution, as is, various enhancements can be added and the streaming video protocol refined.

The following improvements can be considered

- The DVR can be improved to provide analogue inputs and outputs and be used in conjunction with the cellphone as a home automation solution, apart from security.
- By adding Bluetooth functionality to the DVR and the cellphone application, Bluetooth enabled cellphones could automatically use Bluetooth for communication between the phone and the DVR, when in range of the DVR's Bluetooth device. This can also provide added functionality to a home automation setup, to control automatic gates, lights and even multimedia devices in the home.
- The security of the DVR can be further improved, by using advanced encryption algorithms for the authentication process.
- The setup process when adding new cameras to the DVR can be automated. The setup of the cellphone application can also be automated, by providing a central Internet database, from which the cellphone can retrieve its DVR address and authentication settings. This requires the DVR to periodically upload new settings to the database.
- A database of compatible cellphones should be compiled, as cellphones are tested with the applications.

Apart from the general improvements mentioned, the biggest challenge is to stay in line with new technologies that are introduced. There is currently a rapid increase in the amount of multimedia enabled handsets and a decrease in the amount of older phones in use. The capabilities of the newer handsets open up various new possibilities to improve the video streaming and even to implement conventional RTP-based streaming in addition to the streaming developed in this Thesis. By doing this, the chance of losing functionality and quality, in order to support older technology, is reduced.

As in any software development project, the amount of future work and improvements that can be undertaken is endless.

REFERENCES

[1] Howstuffworks.com information resource, founded by North Carolina State University, since 1998
www.howstuffworks.com

[2] Wikipedia free Encyclopedia, WikiMedia Foundation, Since 2001, *WikiPedia OSI Model*
www.wikipedia.org

[3] Cisco Systems, Inc, *Internetworking Technology Overview*, June 1999
www.cisco.com

[4] Carlo Fonda, Marco Zennaro, The International Centre for Theoretical Physics, *Networking Basics*,

[5] Petery Rysavy, Rysavy Research, *Data Capabilities, GPRS to HSDPA, Sept 2004*

[6] Usha Communications Technology, *General Packet Radio Service, 2000*

[7] Petery Rysavy, Rysavy Research, *Mobile Broadband, September 2006*

[8] Malek Sraj, Masters Degree Project, *EDGE Downlink Throughput Performance, 2006*

[9] Internet RFC/STD/FYI/BCP Archives, *RFC 768 – User Datagram Protocol*
www.rfc-archive.org

[10] Yaniv Pessach, Microsoft Corporation, *The pros and cons of TCP and UDP*

[11] Internet RFC/STD/FYI/BCP Archives, *RFC 793 – Transmission Control Protocol*
www.rfc-archive.org

[12] Keith W. Ross, James F. Kurose, *Connectionless Transport: UDP, 2000*

[13] Internet RFC/STD/FYI/BCP Archives , *RFC 1880 – Real-time Transfer Protocol*,
www.rfc-archive.org

[14] Colin Perkins, USC Information Sciences Institute, *RTP: Multimedia Streaming over IP*

[15] Internet RFC/STD/FYI/BCP Archives , *RFC 2616 – Hypertext Transfer Protocol*
www.rfc-archive.org

[16] The WAVE report, *Video Compression Technology*
www.wave-report.com

[17] Nokia Mobiles Phones, *Multimedia Streaming White Paper*

[18] Juka Helin, MediaLab, Mobile Device Developments, *Streaming and Broadcasting Services for Mobile Handsets, Sept 2004*

[19] TeliaSonera GPRS Content Conference, *Insights into Lessons Learned with Video Services over GPRS Networks, Oct 2003*

[20] Rob Koenen, Intertrust Technologies,Apple Worldwide Development Conference, *MPEG-4 Demystified, Jun 2003*

[21] Wikipedia free Encyclopedia, WikiMedia Foundation, Since 2001, *WikiPedia – MPEG-4*
www.wikipedia.org

[22] Riley and Richardson, Artech House, *Digital Video Communications*

[23] Sun Microsystems
http://java.sun.com/

[24] Srikanth Raju, Sun Microsystems, *Developing Wireless Applications in Java using Java Micro Edition (J2ME)*

[25] Ferdinand Alimadhi, *Wireless Access to Web-based Interfaces of Legacy Simulations, 2002*

[26] SearchWeb Services
www.searchwebservices.com

[27] JSR 118 Expert Group, *Mobile Information Device Profile for Java 2 Micro Edition, version 2.1*

[28] Sun Microsystems, *Mobile Information Device Profile, JCP Specification, Java 2 Micro Edition, 1.0a*

[29] MyADSL
www.mybroadband.co.za

[30] Zhen Wen, Zicheng Liu, Michael Cohen, Jin Le, Ke Zheng, Tomas Huang, *Low Bit-rate Video Streaming for Face-to-face Teleconference*

[31] Wikipedia free Encyclopedia, WikiMedia Foundation, Since 2001, *WikiPedia - Delphi*

[32] Nokia Developer's Discussion Forum

http://discussion.forum.nokia.com

[33] Wayne Forrest, Stich Technologies

[34] Torry's Delphi Pages
www.torry.net