Report

SymPhone
Design and implementation of a VoIP peer for Symbian mobile phones using Bluetooth and SIP

Author(s):
Studi, Patrick; Frei, Andreas; Burdet, Luc; Alonso, Gustavo

Publication Date:
2011

Permanent Link:
https://doi.org/10.3929/ethz-a-006780609 →

SymPhone: Design and Implementation of a VoIP peer for Symbian mobile phones using Bluetooth and SIP

Patrick Stuedi, Andreas Frei, Luc Burdet and Gustavo Alonso*
Swiss Federal Institute of Technology (ETHZ)
Departement of Computer Science
8092, ETH-Zentrum, Switzerland

Abstract

VoIP is born from the growing Internet infrastructure, which has over the years seen significant improvements in both bandwidth and end-to-end latency. In this paper, we explore making VoIP available on a mobile phone. For that purpose, we propose an architecture and describe the various components involved. Data entering and leaving the mobile phone is encapsulated in a wireless Bluetooth connection. The bridge to the Internet is provided by a linux Bluetooth access point. The system is compatible with current VoIP standards using RTP and SIP for data and signaling transmission, respectively. It has been tested to work in combination with several known softphones. Both RTP and SIP stack are built on top of a special IP emulation layer on the mobile phone, which has been developed to facilitate the migration of the application onto devices having a direct IP binding available for Bluetooth such as BNEP (Bluetooth Network Encapsulation Protocol). Apart from presenting design and implementation details, the paper also provides measurement results with regard to delay and session setup time and discusses certain limitations of the application evolving from restrictions imposed by the mobile phone's programming platform.

1 Introduction

According to recent reports, a high percentage of mobile phone calls occur at home or at the workplace. These calls are often performed within meters of an existing well-established and less expensive telecommunications infrastructure. Such is the price to pay for the convenience of being "always-available". A major focus for research in recent years by telecommunications companies has been in finding marketable ways to solve this by merging telephony technologies and provide better, less-expensive, services to customers [4, 3, 9]. Merging VoIP technologies with telephony infrastructures is of particular interest since it may significantly reduce the costs.

In this paper, we explore making Voice over IP (VoIP) available on a mobile phone using Bluetooth as the access protocol. Bluetooth was selected because it is increasingly available in mobile telephones. Most modern mobile phones with a focus on wirelessly sharing data between the device and a host PC come equipped with a Bluetooth adaptor.

Since an important goal was to implement a prototype that runs on a broad spectrum of devices, SymbianOS [5] was chosen to be the target platform for the mobile phone. Symbian is an operating system for mobile devices widely used in industry. It is designed for the specific requirements of advanced 2G, 2.5G and 3G mobile phones and provides a native C++ application programming interface.

Our system uses an out-of-the-box Linux PC to serve as an Internet access point for both voice and signaling data. When in range of a PC, a wireless Bluetooth connection to the IP network is made available to the mobile phone, offering the choice of connecting to a conventional mobile GSM network or to a lower-cost IP infrastructure for VoIP. The vision for the future is to make the mobile phone's operation fully transparent to the user by making both technologies completely interchangeable (Figure 1).

In this paper, we discuss how the Bluetooth connection to a mobile phone can be used to route telephony data. The solution we provide is based on current standards for call setup and media transmission and therefore allows the mobile phone to setup phone calls with most of the currently available soft-phones[1]. It is also possible to setup a VoIP call over the internet between to Symbian mobile phones using our application.

The main contribution of the paper is a generic architecture for Bluetooth-based VoIP clients running on Symbian-based end-user devices, such as mobile phones. The paper also provides insight into concrete issues that arise when actually bridging the gap between the architecture and the real implementation.

The remainder of this paper is structured as follows. We present in section 2 the architecture of the system, putting

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems NCCR-MICS. A center supported by the Swiss National Science Foundation under grant number 5005-67322.

[1] Kphone [6],Linphone [7],Twinkle [8]

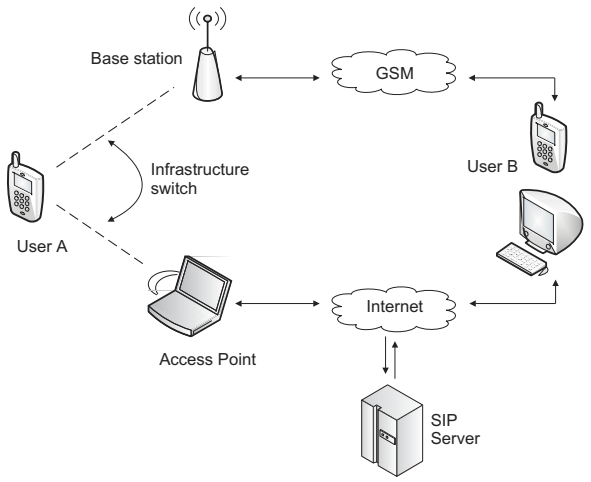

Figure 1. VoIP infrastructure

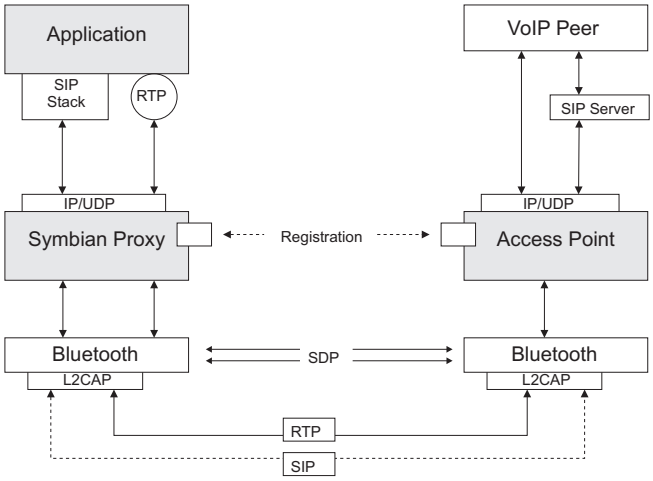

Figure 2. Architecture

into context the main components necessary to achieve our goal. Section 3 discusses the concept of tunnelling IP data through a Bluetooth connection from the mobile phone to a Linux access point. In section 4.1 we give some detailed information on how SIP signaling can be used to establish calls into the Internet. The voice engine recording and playing audio streams is described in section 4.2. Section 5 shows some measurement results. A short review of related work is given in section 6 and section 7 concludes the paper.

2 System Architecture

2.1 Background

VoIP applications typically comply with several protocol standards. This is, either H.323 [12] or SIP [21] for setting up the call, and RTP/RSCP [18] for realtime audio data exchange. After a long and at times heated debate about what call setup protocol is best suited to replace ISDN's SS7 infrastructure, the industry has now settled on the Session Initiation Protocol (SIP), an IETF standard. While most Voice over IP (VoIP) systems deployed in the enterprise as a replacement of aging PBX systems still remain proprietary, carriers and ISPs have started to invest heavily in their next generation infrastructure based on SIP. In order to stay compatible with this trend, we decided to also use SIP as the signaling protocol for the VoIP application, together with RTP/RTCP for audio communication. Compatibility with current standards certainly is an absolute must to allow the system to be used in combination with already available VoIP clients.

While the protocol interface is relatively straightforward, its implementation is quite challenging because of restrictions in the Symbian operating system:

1. A specific feature of the Bluetooth specification [13] is BNEP [11](Bluetooth Network Encapsulation Protocol). BNEP allows IP traffic to be transmitted over a Bluetooth L2CAP connection. Unfortunately, BNEP is not yet available on SymbianOS 7.0, which is the version we have used for this work.

2. According to the Nokia 6600 specification, a maximum of one Bluetooth connection at a time may be active at the mobile device.

3. The SymbianOS streaming API allows for streaming audio data with variable granularity in the buffer size. However, the SymbianOS Multimedia Framework (MMF) does not use the specified buffer size directly for sampling data, but fills the variable size application buffers from an internal 700ms buffer.

The first point requires some sort of IP emulator to be implemented on the mobile phone in order to support sending and receiving of SIP and RTP packets. The second point requires the mobile phone to multiplex several data connection onto one single Bluetooth channel. The next section shows how our design addresses these two restrictions. The third point imposes a firm lower bound to the delay of the VoIP application, as we will see in section 5 when we discuss experimental results.

2.2 Main components

The architecture of our VoIP system is divided into three main parts (Figure 2): The *Symbian proxy*, the *Bluetooth access point*, and the actual VoIP application.

The Symbian proxy provides transparent IP connections to remote destinations, hiding all the lower layer Bluetooth functionality. Applications create new IP tunnel endpoints by using a dedicated register interface. After having created such an endpoint, applications can use regular sockets to communicate with the outside world. Besides IP emulation, the proxy is also responsible for multiplexing different

IP connections onto the single Bluetooth channel. The multiplexed data stream is later de-multiplexed on the access point for further processing.

The Bluetooth-connected access point serves as a dual-interface router for all IP packets between the mobile phone and the target destination(s). While the Bluetooth interface offers a simple wireless connection to the mobile device, the access point's IP-connected interface will be the component communicating directly with the mobile phone's intended peers, namely the SIP server in the case of a VoIP call setup and the VoIP client in the case of RTP data transmission. In order to be easily identified by the mobile phone, the access point publishes an SDP [13] service advertisement helping the *Symbian proxy* to dynamically look up the access point's location and establish a connection to it.

The VoIP application sits on top of the *Symbian Proxy* using the Session Initiation Protocol (SIP) to setup calls with other peers and RTP for audio communication. At startup, the application creates two IP endpoints for both the SIP and the RTP channel using the *Symbian Proxy*. The Application is responsible for setting up the call and interaction with the Symbian Multimedia Framework that provides access to the microphone and the loudspeakers. Instead of including Bluetooth functionality directly into a VoIP client, we opted for separating functionality and stacking them one atop another. This facilitates the migration of the application onto devices having a direct IP binding available for Bluetooth.

In the following sections we explore each of these components in more detail.

3 Bluetooth IP Tunnel

3.1 Protocols and Registration mechanism

While the *Symbian Proxy* provides transparent IP access and multiplexes different IP streams onto Bluetooth, the *Bluetooth access point* (Figure 2) is in charge of demultiplexing these streams for further transmission into the Internet. Both parts build a so called Bluetooth IP tunnel. In order for the tunnel to provide the multiplexing/demultiplexing capability, a common header is used for transmission over a Bluetooth L2CAP connection (see Table 1). The *ThreadID* allows to properly forward the packets to their intended tunnel endpoint and final destination. Note that this packet format is not reserved for Bluetooth exclusively, but could be applied to other forms of wireless (or wired) links to the mobile phone. The common header was kept as small as possible, so as to keep its relative size insignificant when compared to the size of the actual packets sent over the connection. The current implementation is not optimal because it includes non-essential payload length and synchronization sequence fields, which are, however needed when building an IP tunnel on top of the Bluetooth RFComm layer. The header format is further designed to support both control and data channels using the *HdrType*

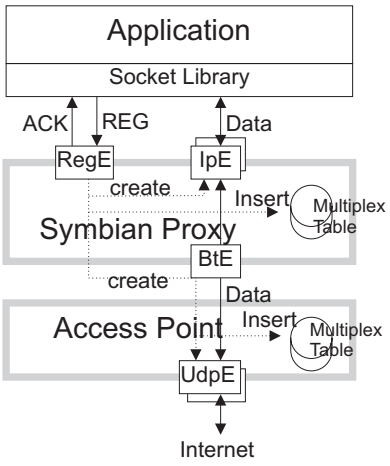

Figure 3. Symbian Proxy and IP tunnel endpoint registration

field. Data packets may contain SIP or RTP packets. Control packets on the other hand, are used internally to signal the creation of new IP endpoints. As mentioned, the proxy allows to create IP endpoints through a register interface. Signaling these registrations to both proxy and access point is necessary since both parts have to agree on a certain mapping between the endpoint socket and the corresponding *ThreadID*. For this purpose, a special control message is sent over an IP tunnel control channel (*HdrType*=REG). The format of such a message is shown in Table 2. A registration message has to be sent to the access point by the proxy every time a new endpoint is created. The proxy binds locally a so called *IP engine* (IpE) to the application's intended destination port, and inserts a special association (application-id, IpE) into its own internal multiplex/demultiplex construct (multiplex table). The access point uses the registration message's contents (source and destination addresses, ports) to establish a communication channel with the intended target of each message by binding locally to the provided source port, and sending to the intended destination address and ports. Figure 3 illustrates the registration process. After a registration is correctly set up at both ends, the local binding information is accessed (from the multiplex table) upon packet arrival to determine (based on the packet's header *ThreadID*) which tunnel endpoint the data must be forwarded to. From the application point of view, the whole tunnelling mechanism is fully transparent due to a special socket library hiding all the proxy registration calls. For migrating the application to a BNEP-capable device, only the socket library has to be replaced.

3.2 Symbian Proxy

From an operating system point of view, the Symbian proxy is a stand-alone application implementing a server process. It is centered around a so called "*engine*" princi-

Offset	Field	Type	Length	Description
0	ThreadId	int16	2 bytes	the unique ID of endpoint and application data is for/from
2	HdrType	int16	2 bytes	the type of data being transported in this packet: 0x01=REG message, 0x02=normal data
4	PayloadLength	int32	4 bytes	the length of the payload in number of bytes
8	Separator	char	4*1 bytes	end of header (synchronisation) sequence '++++'

Table 1. Packet format for IP data tunneling (common header).

Offset	Field	Type	Length	Description
0	Id	int16	2 bytes	a unique ID for each application, used for mux/demux
2	SrcPort	int16	2 bytes	the IP port to bind to at the Access_Point
4	DestAddr	uint32	4 bytes	the remote IP address for UDP packets (send to and receive from)
8	DestPort	int16	2 bytes	the remote IP port for UDP packets (send to and receive from)
10	DataType	int16	2 bytes	the type of data being tunnelled: 0x01=binary, 0x02=ASCII

Table 2. Registration message format.

ple. Namely, the service consists of a central coordination engine and a series of satellite engines to do all the actual work (Figure 3).

The coordination engine implements a state machine (Figure 4) which keeps track of the connection status of all sub-engines: the registration engine, the Bluetooth engine and all the IP engines representing tunnel endpoints. The current state of the proxy is an indicator of the readiness level for tunnelling data from connecting applications. When unconnected, the proxy receives 'Connect' commands from the user interface, and initiates connection attempts in both the registration engine (regE) and the Bluetooth engine (btE), which in turn initiate internal connect procedures depending on their current state. The *Symphone* application may start initiating a VoIP call setup as soon as the proxy's state is switched to 'Connected'.

The **Bluetooth engine** (btE) is in charge of communicating with a remote peer over a wireless Bluetooth link. When asked to 'Connect', *btE* initiates a search to seek out a suitable peer using Bluetooth SDP [13] discovery queries. It queries for a custom 'AccessPoint' service which the Bluetooth access point defines upon start, as will be explained in section 3.3. The Bluetooth engine further provides *read* and *write* functionality to transmit and receive packets to and from the Bluetooth link. Incoming packets are stripped off the tunnelling header and passed on to the corresponding IP engine (IpE), based on the *ThreadID* and the local binding for that ID.

The **registration engine** (regE) represents an external interface made available to applications for signalling intent to create a new IP tunnel endpoint. Upon reception of a registration request, the engine notifies the proxy's coordination engine, which in turn forwards the request over the

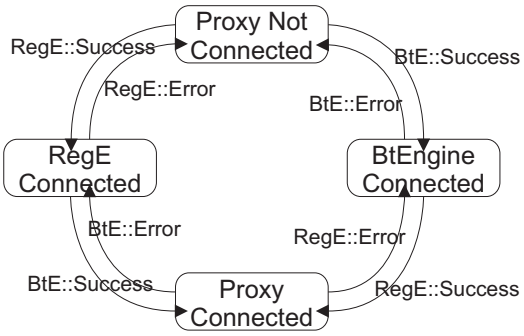

Figure 4. The proxy's internal state machine

(connected) Bluetooth engine and creates an instance *IpE* using the remote port provided in the request.

The **IP engine** (IpE) communicates with the application on the mobile phone by abstracting the underlying IP medium. However, the application does not interact with the **IpE** directly. It uses a special socket library as described in section 3.1. The **IP engine** does little more than shuffling application data down the Bluetooth engine and vice versa.

3.3 Access Point

As mentioned, the Bluetooth access point acts as a bridge between the wireless Bluetooth link and the Internet. The first important role of the access point is to advertise to passing-by mobile phones the service it provides. It does so by using the Bluetooth Service Discovery Protocol (SDP) [10]. In SDP, a dynamic network of Bluetooth devices may browse remote devices for their services, and properties, allowing for better network dynamism and re-

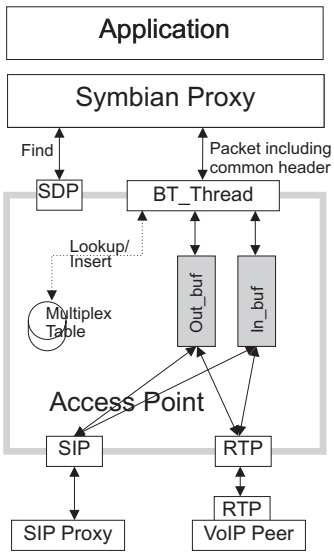

Figure 5. Internal structure of the access point

placing the need for a central repository. SDP services are characterized by their service class and their service record (a set of attributes). The access point acts as an SDP server for a new service class. It publishes a service record for the "A_P" service with the most important characteristics listed in Table 3.

Internally, the access point implements a traditional multi-threaded server process. A **bt_listener** creates new **bt_reader** and **bt_writer** threads upon connection attempt. The threads are then responsible for processing registration and data messages. Using the application ID of the message, a bt_reader creates or updates a new tunnel endpoint. This is done by creating two new **ip_udp_reader** and **ip_udp_writer** threads, looking up the IP address and port information in the registration message, and subsequently inserting the corresponding association (application-id, thread) into an internal dictionary structure (multiplex table). For regular data, bt_reader is a producer over the synchronized **out_buf** data buffer, notifying the ip_udp_writer thread specified by the tunnelling header. The ip_udp_reader and ip_udp_writer are producer and consumer threads over the **out_buf** and **in_buf** synchronized buffers, respectively. For an illustration on the access point's internal structure see Figure 5.

4 SymPhone Application

SymPhone is a multi-view Symbian Series 60 application [22, 20]. It uses the underlying Symbian proxy for transparent IP communication over Bluetooth. Its internal structure can be subdivided into two parts: a SIP signaling engine and an audio engine (Figure 6).

4.1 Signaling Engine

The SIP engine takes all the steps necessary to initiate or tear-down SIP sessions with a remote peer. It handles incoming requests and propagates decisions to the GUI only when user interaction is necessary (pick-up or hang up a call). One issue with the SIP engine is that it could not be built on the Nokia SIP stack since the library is based entirely on the use of Internet Access Providers (IAPs). On Symbian mobile phones, the only IAPs available are stored in the system-wide storage for communication-related settings and refer to the GSM network interface for IP. Without providing a specific IAP, the Nokia SIP stack never receives a valid IP address. Unfortunately, the main part of the SIP library is closed, so simply recompiling it for a different transport mechanism is not possible. The solution that we chose was to re-use some of the existing structure and objects of the library to re-implement the communication functionality needed. This facilitates a potential replacement of the SIP stack with an official Bluetooth capable SIP library at a later point in time. The new SIPSocketStack we provide is a very basic SIP message parser and supports the sending of REGISTER, INVITE, ACK and BYE requests. These messages correspond to the four basic SIP protocol primitives needed for a SIP call setup.

4.2 Audio Engine

The audio engine in the SymPhone application implements a VoIP specific use of the Symbian Multimedia Framework (MMF) [19]. The engine processes incoming RTP audio streams and feeds raw audio data into the loudspeaker. It also reads consecutive audio data from microphone and transmits RTP streams to the remote peer. Raw audio is referred to as uncompressed, 16-bit PCM16 encoded data. Audio data entering and leaving the network is 8-bit encoded and has to be transformed at reception and transmission respectively. Due to the lack of a Symbian RTP library, a subset of such a library has been implemented as part of this work. The library stays true to the specification when it comes to RTP headers, but leaving out completely such functionalities as packet counting and reordering. Outgoing packets will be correctly labelled for the peer's RTP stack, but the headers of incoming packets are stripped and ignored. Similar to the SIPSocket stack described in section 4.1, the library approach facilitates the replacement with an official RTP library at a later point in time. The audio engine accepts recorded audio data in a buffer, and splits it according to a pre-determined RTP payload size, which is currently 160 bytes. Each packet starts with a valid header containing a sequence number and a timestamp before being forwarded to the underlying RTP/IP interface. Unfortunately, the Nokia 6600 we are using does not have full-duplex audio capabilities. Symbian specifies that this is because of the 'one-shot' recording or playback design policy opted for by most phone manufacturers [17].

attribute	ID	value
SDP_ATTR_RECORD_HANDLE	0x0	random
SDP_ATTR_SVCLASS_ID_LIST	0x1	ACCESS_POINT_SVCLASS(0x1334), GENERIC_TELEPHONY_SVCLASS_ID
SDP_ATTR_PROTO_DESC_LIST	0x4	L2CAP_UUID
Name	0x100	"Access_Point"
Description	0x101	"Bluetooth-to-IP access point for tunnelling binary&ASCII packets"
Provider	0x102	ETHZ::IKS::LAB

Table 3. Access Point SDP record

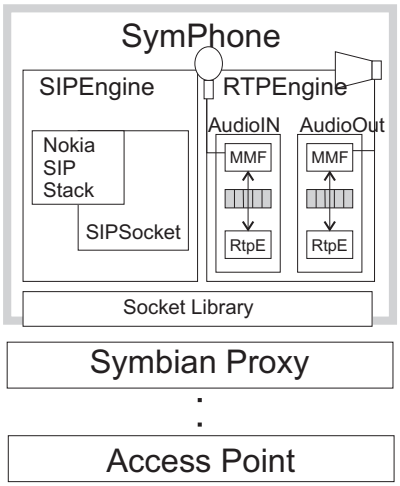

Figure 6. The SymPhone application

The solution taken with this problem was to implement the application in a push-to-talk manner.

5 Evaluation

Several aspects are of interest when testing a VoIP application: setup time, end-to-end latency and packet loss. Furthermore, to objectively measure the speech quality, the ITU PESQ [16] method could be have been employed. However, PESQ measurements are complex and require explicit equipment which we did not have available. Furthermore, PESQ includes many parameters like filtering, distortions, channel errors which can be ignored when evaluating the very generic behavior of an application. We therefore concentrated on SIP setup time and latency measurements.

Our setup consists of a mobile phone, a linux access point, a sip proxy/registrar and a softphone application[2]. To avoid introducing additional delay, the access point, the proxy and the softphone were all located on the same physical machine. After having registered both the softphone and the mobile phone with the sip proxy/registrar, a VoIP call is initiated from the mobile phone to the softphone. In our 10 consecutive experiments, we measured an average *dial-to-ring* latency of about 300ms. The dial-to-ring latency includes the time from the moment the first SIP Invite message is triggered on the mobile phone, until a 180 SIP Ringing message is received. The exact message flow is illustrated in Figure 7.

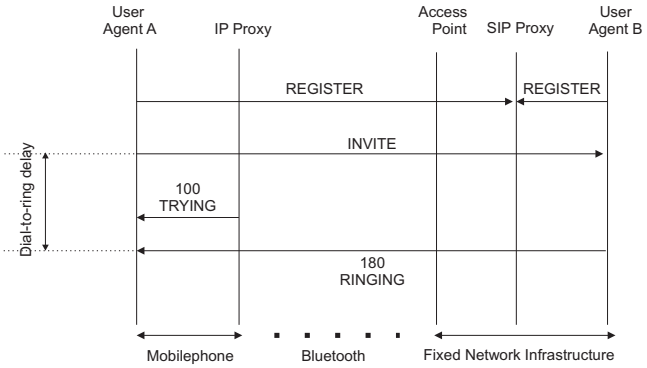

Figure 7. SIP setup message flow

A dial-to-ring latency of 300ms is comparable to the setup time needed for an overseas call in a fixed network infrastructure [15]. The overhead is introduced by the slow Bluetooth connection and the various components (proxy, access point) involved.

For measuring end-to-end latencies we made use of a simple mixer, mixing the sound recorded at the mobile phone's microphone towards the left channel, and the sound received at the softphone's loudspeaker towards the right channel (Figure 8). The final stereo signal was recorded with 48200Hz into a WAV-file, as illustrated in Figure 9. The difference between two consecutive amplitudes refers to the latency between the signal recorded at the mobile phone and the signal received by the softphone. With respect to the sampling frequency of 48200Hz, the latency computes to approximately 770ms, which is far beyond what would be acceptable for interactive voice.

We have investigated the cause of this problem as follows. The Symbian OS' multimedia API providing access to the low level audio devices, like microphone and loudspeakers, contains an internal sample buffer of 11200 bytes. When sampling audio with 8000Hz, it takes 700ms to fill the buffer, regardless of the buffer size used in the audio

[2]We have used Kphone[6]

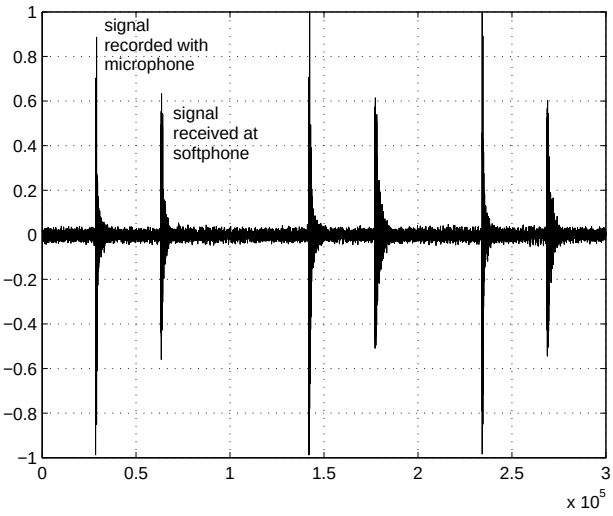

Figure 9. Measured latency

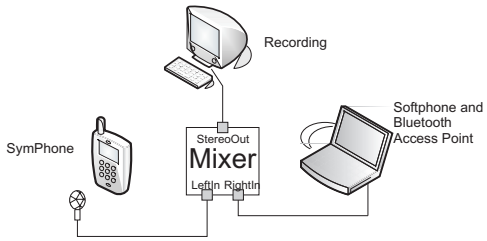

Figure 8. The evaluation setup

Time	**Buffer Size**	
in ms	320 bytes	640 bytes
0	0.9512%	0.9193%
< 15	0.0203%	0.0252%
$\approx$ 700ms	0.0286%	0.0556%

Table 4. Relative frequencies of audio recording callbacks

engine[3]. This behavior has been confirmed in a small experiment, where we only tested the SymbianOS callback function, as shown in Figure 10. We have measured callback interval times for two different buffer sizes, 320 bytes and 620 bytes. In the presence of an internal buffer, the relative frequencies of large 700ms callback latencies can be expected to grow linearly with the size of the application buffer passed to the API. Table 4 shows the result of the experiment. Let us look at the 320 bytes case first. While sampling 320 bytes took $0ms$ in 95.12% of all cases, it took 700ms in 2.86% of the cases. If we neglect the cases where the sampling time was slightly above 0ms ($< 15ms$), we observe that the SymbianOS internal buffer has to be re-filled after every approximately $0.9512/0.0286 \approx 33$ consecutive sampling requests, which results in an internal buffersize of $320 \times 33 = 10560$ bytes. If we pass a buffer of size 640 bytes instead, the internal buffer is re-filled after every $0.9193/0.0556 = 16$ sampling request, hence twice as much, which is what we expected. In the future, one might be able to bypass this problem by using the latest Symbian API for audio sampling. We nevertheless we believe that the design we have presented has acceptable overhead, since once the problem with the buffer is solved, the additional overhead amounts to 70ms. This is more than acceptable and is within the limit of applications supporting interactive voice, even when considering the additional latency from the access point to some peer located in the internet.

[3]The buffer the audio engine passes to the Symbian multimedia API contains space for 160 bytes of raw 16bit PCM audio data

6 Related Work

Although there is a clear trend towards merging the telephony and Internet technologies, we do not know of any SIP based VoIP application on the mobile phone that runs over a low cost Bluetooth connection. In a recent work [1], a VoIP phone was developed running on Symbian OS putting a proprietary non-SIP-based protocol for call establishment. A SIP-based phone communicating over Bluetooth was announced by British Telecom (BT) [14]. However, their approach includes special equipment like a BT access point and a customized mobile phone. The idea of tunneling IP based communication using Bluetooth has also been adopted by GnuBox [2], an open source project of the Symbian OS Communitiy.

```
void CDeviceIoMmfEngine::Read(){
    startTimer()
    iInputStream = CMdaAudioInputStream::NewL(...);
    // read a buffer from the audio input stream
    // upon completion we will receive a callback in MaiscBufferCopied()
    iInputStream->ReadL(iStreamBuffer[iStreamIdx]->Data());
    // start a timer to measure latency
    startTimer()
}
...
void CDeviceIoMmfEngine::MaiscBufferCopied(TInt aError, const TDesC8& aBuffer*){
    // measure latency of callback
    stopTimer()
    Read()
}
```

Figure 10. Measuring the latency of the Symbian MMF API

7 Conclusion

In this paper we presented the design and architecture of a VoIP client on a Symbian mobile phone. We have described how the lower-cost Bluetooth connection to a mobile phone can be used to route telephony data. The solution we provide is based on current standards for call setup and media transmission and therefore allows the Symbian mobile phone to setup phone calls with most of the currently available softphones. In the paper we also provide detailed information about software issues and restrictions related to the Symbian OS audio interface. The most important restriction was found in the Multimedia Framework provided by Nokia's Symbian OS, which causes a large sample delay due to the large internal buffer. In our evaluation we have shown however, that the end-to-end delay does not significantly exceed the sample time. Another issue is that current versions of Symbian OS do not allow multimedia devices to be read/written in full-duplex mode. Once these issues are addressed by later versions of the Symbian API and more flexible hardware, the design we propose provides a viable solution to connect mobile phones to a VoIP network using Bluetooth as the access protocol.

References

[1] *Bluesox, bridging distances*. http://bluesox.eidelen.ch.

[2] *GnuBox*. http://gnubox.dnsalias.org/gnubox/.

[3] *My mobile phone is also a fixed network*. http://www.teltarif.de/arch/2005/kw24/s17460.

[4] *Net2Phone VoiceLine XJ100 Wi-Fi VoIP Phone*. http://www.mobilemag.com/content/100/340/C3309.

[5] *The Symbian Operating System*. http://www.symbian.org.

[6] *The Symbian Operating System*. http://www.wirlab.net/kphone.

[7] *Telephony on linux*. http://www.linphone.org/.

[8] *Telephony on linux*. http://www.twinklephone.com.

[9] *The WIreless Directory. Bluetooth enterprise access infrastructure*. http://www.thewirelessdirectory.com/Bluetooh-Product/Bluetooth-Enterprise-Access-Infrastructure.htm.

[10] *Specification of the Bluetooth System*. http://www.bluetooth.com, 1999.

[11] Bluetooth Network Encapsulation Protocol, June 2001. http://grouper.ieee.org/groups/802/15/Bluetooth/BNEP.pdf.

[12] *Packet-based multimedia communications systems*. http://www.itu.int, 2003.

[13] Specification of the Bluetooth System, Core Package version 1.2, Nov. 2003. https://www.bluetooth.org/spec/.

[14] British Telecom, http://www.bt.com. *Bluephone*.

[15] I. Curcio and M. Lundan. Sip call setup delay in 3g networks. In *ISCC'02: Proceedings of the Seventh International Symposium on Computers and Communications*, 2002.

[16] N. M. For. Perceptual evaluation of speech quality (pesq) – a.

[17] Forum Nokia. *Developer Platform 2.0: Known Issues v2.0*, May 2005.

[18] A.-V. T. W. Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889 (Proposed Standard), Jan. 1996. Obsoleted by RFC 3550.

[19] N. Johnson. *Audio Streaming: How to successfully stream audio on Symbian OS v7.0s v1.1*. Symbian Ltd., July 2004.

[20] Nokia Corporation. *Developer Platform 2.0 for Series 60: Application Framework Handbook v1.0*, December 2003.

[21] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853.

[22] Symbian Ltd., http://www.forum.nokia.com/main. *Symbian OS Series 60 Service Development Kit for 2nd Edition*.