Technical University Berlin

Telecommunication Networks Group

Improving Wireless Internet Access combining an Active Network Approach and a Proxy Architecture

Morten Schläger, Andreas Willig

{morten,willig}@ft.ee.tu-berlin.de

Berlin, May 1999

TKN Technical Report TKN-99-003

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Abstract

In this paper we tackle the problem of providing wireless internet access with good performance and over heterogeneous networks. We propose to combine two approaches: our Remote Socket Architecture is a proxy approach on the transport layer splitting the socket interface between mobile and base station, while our Architecture for dynamic switching of so-called last hop protocols is an active network approach. The basic idea of the latter is to download lower layer protocols dynamically from the base station to the mobiles in such a manner, that a reliable service without losses and duplicates is guaranteed, even when protocol switching is abrupt.

Contents

Chapter 1

Introduction

There is currently an enormous growing interest in providing internet access not only to fixed users, but also to mobile users anywhere and anytime using wireless technologies ("Access is the killer app" [18]). Users want to be able to move freely and do their work, e.g. web browsing, transferring files using ftp, remote login at their employer and so forth. From a protocol point of view these main internet applications most often use TCP on top of IP. Every approach for wireless internet access must support this. But unfortunately there are two problems: first, TCP is known to have serious performance problems with wireless links [8], [4], and second, the designer is faced with a large number of greatly different wireless technologies, e.g. DECT, GSM, Wireless LANs, Infrared, ..., each having distinct characteristics and transmission ranges. In an ideal world the user does not know about the underlying technology, it is transparent to him, even when moving. The problems with heterogenity become even more serious, if the end system is a small device, like e.g. a personal digital assistant (PDA). Instead of requiring each terminal to have available all needed network adapters and protocol software, it is very attractive to employ only a single network adapter and to download the necessary software. For transition between different transmission technologies this approach gives rise to "software radio" [3], for transition between different protocols the concept of "Active Networks" comes in handy [7] [17]

In order to attack the performance problem as well as the heterogenity problem we find it important to jointly attack both problems by combining two design approaches:

- A transport-layer proxy approach (called *Remote Socket Architecture*, ReSoA)
- and an active network approach, which is used for dynamically downloading of lower layer protocols.

Both proxy architectures and active network architectures are a suitable way to cope with heterogenity and for providing internet access especially to small handheld devices, additionally they have performance benefits especially for TCP connections. Instead of a PDA having several different interfaces for different wireless technologies, it is more convenient to download the necessary protocol software on the fly from the network to the mobile device.

An architecture for dynamic downloading of protocols allows not only to change between different wireless technologies, but also for using different protocols over the same technology[1]. This may be helpful for internet service providers (ISPs) to offer specifically tailored access protocols and thus to differentiate itself from other providers without losing compatibility. Furthermore, even for a single provider and a single technology it offers the possibility to dynamically adapt the protocol in use to changing wireless link characteristics.

The challenge is not only the pure downloading of protocol software, but rather to allow seamless switching of protocols while maintaining a reliable service without losses and duplicates, which is needed by the remote socket architecture. Furthermore we assume that it is necessary to perform abrupt protocol switching, since when moving fast and roaming e.g. between DECT and GSM there will be no time for graceful protocol termination.

In this paper we do mainly three things. First we give a motivation and a short overview on the remote socket architecture as a special case of a proxy architecture (section 2). Then we analyze the problems which occur when protocols are switched dynamically and abrupt (section 3). And third we develop a software and protocol architecture which allows for seamless switching while maintaining the reliable service (section 4).

[1]The idea of switching on the fly between different technologies and protocols is not only useful in the internet world, but also in the ATM world. An example of such an architecture is described in [6].

Chapter 2

Remote Socket Architecture for Wireless Internet Access

With the advent of small handheld devices, wireless access and mobility a lot of new requirements are set on the provision of internet access. The introduction of proxy architectures is a suitable way to cope with these requirements. In general, proxy architectures are a way of distributing complexity and protocol or application intelligence, such that most of the work is done in the inner network elements, not in the small end devices. One example of application layer proxies is given in [9] (which is part of the BARWAN project described in [5] [11]), however we are more interested in transport layer proxies, like e.g. MOWGLI [12] and the *Remote Socket Architecture* (ReSoA), which we describe now.

First we describe the basic scenario which we assume, see fig. 2.1. Consider a user with a small handheld device (e.g. a personal digital assistant, PDA) or a laptop. The user wants to access the internet services, thus using a set of TCP connections, mostly with fixed hosts in the internet. In virtually all cases the wireless hop is the last hop of a TCP connection, all other hops and the remote end point being located in the fixed internet (we do not cover the case of a TCP connection between two PDAs within the same location).

2.1 Basic Idea of Remote Socket Architecture

Consider the scenario as described above. The basic idea behind the Remote Socket Architecture (ReSoA) is to move the socket interface and the TCP/IP protocol stack to the base-station and leave only the user interface (the standard socket calls like *socket*, *connect*, *read*, ..) on the wireless end-system as depicted in fig. 2.2. The TCP applications can use

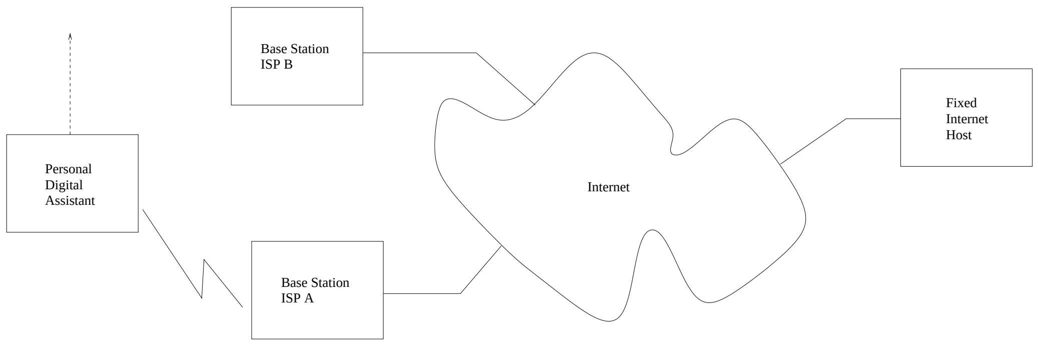

Figure 2.1: Scenario of Usage

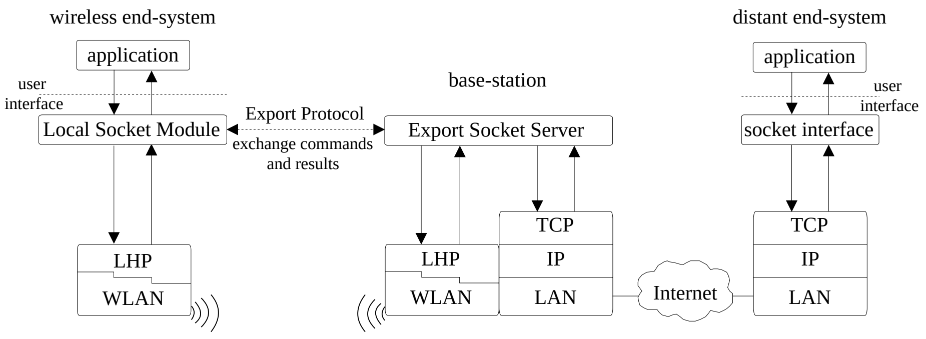

Figure 2.2: Remote Socket Architecture

the socket calls as usual. Application calls to the socket interface are transferred to the base-station by the *local socket module* using the reliable service of the *Last Hop Protocol* (LHP). The corresponding answers are sent by the export socket server back to the local socket module, also using the LHP. The last hop protocol is a tailored and lightweight protocol. The base-station acts as a proxy for the wireless end-systems. We call the proxy-server *export socket server*. The protocol between the export socket server and the local socket module is called *export protocol*. This approach preserves the usual TCP semantics [15].

The design of the export protocol linking the local socket module and the export socket server is independent from the underlying technology. The design of the last hop protocol is dependent on the underlying technology (Wireless LAN, GSM, DECT, ...), the ISP in use, the position and so forth.

A detailed description of the implementation of the socket interface can be found in [19]. A more detailed description of the socket interface usage can be found in[16].

2.2 Service of LHP

The LHP must provide a reliable service to the export protocol. Reliable means that every packet is delivered exactly once in the correct order to the receiver. Unrecoverable errors (e.g. maximum number of retransmission reached, link failure) must be signalled to the service user.

The LHP has to provide three service primitives. Socket calls or responses are delivered over the wireless link using the *data request* primitive. The reception of packets is signalled to the receiving instance of the export protocol using the *data indication* primitives.

The service user can assume a successful delivery of its data request. Therefore positive confirmations are not needed. If the LHP is unable to deliver a packet in spite of its error control mechanisms (e.g. long error bursts) then it generates *link error* primitive, which can be mapped uniquely to the corresponding request primitive.

2.3 Export Protocol

The export protocol is responsible for the export of the interface between the base-station and the wireless end-systems. The main task of the export protocol is to maintain the semantics of the socket interface. It assumes that the LHP provides a reliable data transfer service.

The export protocol is based on a request-response mechanism. The decision to use a request-response protocol follows the typical operation of the socket interface, where an application performs a socket function call which is passed to the socket interface (request) and a result code is returned to the application (response).

The export protocol on the wireless end-system encapsulates the socket function calls (marshalling) and transmits them to the export socket server. After that the export protocol waits for the corresponding response. During this time no other request related to this socket can be sent to the export socket server. When it receives the response it decapsulates the contained result and passes it to the application. The same request-response protocol is used in the direction from the export socket server to the local socket module. The export socket server transmits data received from TCP to the local socket module using a Request. It should be noted that although for a single socket a request-response scheme is used, it is possible to operate several sockets in parallel.

Due to the usage of a reliable last hop protocol the export protocol only needs to send every request and response exactly once. A flow control mechanism is needed in order to prevent socket buffer overflow. To prevent buffer overflow the local socket module and the export socket server signal each other the amount of free buffer space.

2.4 First Results on Remote Socket Architecture

Introducing the remote socket architecture has some important advantages. First, the processing burden and software complexity of the mobile devices is reduced, since only a lightweight protocol needs to be run instead of a full-blown TCP/IP protocol stack. This is especially valuable for small devices like PDAs. The second advantage is the improved TCP performance.

In order to give substance to the latter claim, we briefly present some measurement results taken at our institute, where the standard deviation of ftp throughput at different positions on a single floor is shown, see fig. 2.3. We performed the measurement using a TCP connection between a fixed host in our institute ethernet and a mobile station, using WaveLAN. The base station is fixed (in room R1) and the position of the mobile is varied between different ftp transfers. The ftp data was transferred from the fixed host to the mobile. The layout of the floor is shown in fig. 2.4. From the measurements we define three types of positions: at *good* positions a TCP throughput of more than 100 kBytes/sec is achieved, at *bad* positions 30-100 kBytes/sec and if the throughput is below 30 kBytes/sec we have an *unacceptable* position.

We have compared the throughput between pure TCP and the remote socket architecture with Send and Wait as last hop protocol. We have selected three positions: one good position, one bad position and one unacceptable position. The results are shown in table 2.1. A more detailed description of the measurement setup and the results can be found in [14]

From the results we can conclude two things: first the use of the remote socket architecture is by far superior to raw TCP on bad and unacceptable positions. Second, raw TCP has significantly higher throughput on the good position. This is mainly due to the fact that the last hop protocol we used acknowledges every single frame, while TCP can use cumulative acknowledgements. As a result, the mobile station tends to send fewer acknowledgement frames, leading to fewer collisions on the WaveLAN and saving bandwidth. Thus we can conclude, that even for a single type of wireless technology it can greatly improve performance to use different last hop protocols, depending on the current state of the wireless link.

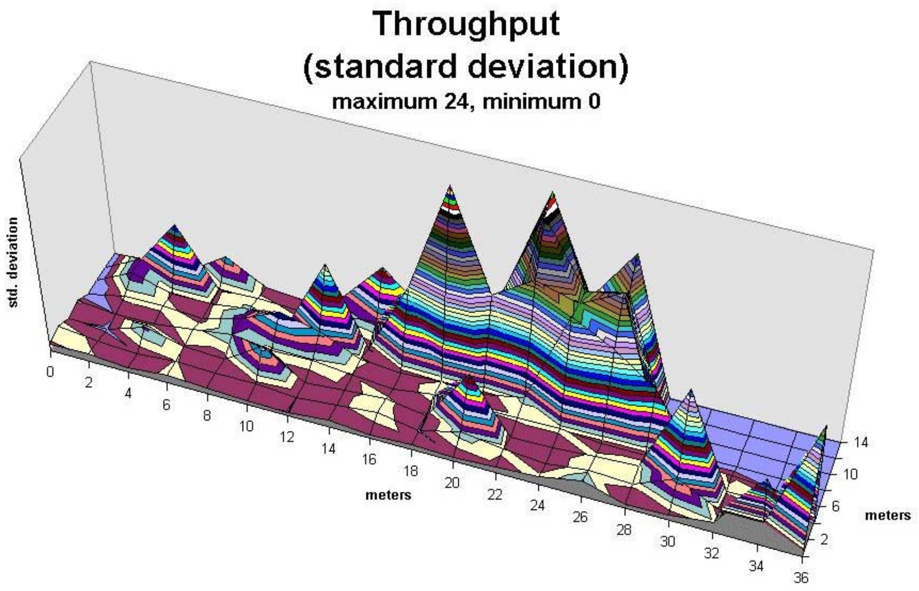

Figure 2.3: TCP throughput

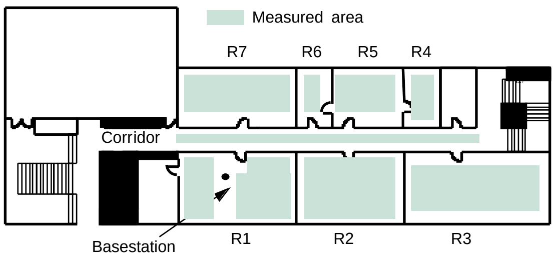

Figure 2.4: Layout of the floor

		Throughput kbyte/s.	
		Mean	std. dev.
Position_A (Good)	TCP	182.2	13.4
	ReSoA	121	3.6
Position_B (Bad)	TCP	77.9	59.7
	ReSoA	122.4	1.7
Position_C (unacceptable)	TCP	-	-
	ReSoA	27.3	8.6

Table 2.1: Throughput characteristics of WaveLan

Chapter 3

Dynamic Switching of Last Hop Protocols

To facilitate the use of different last hop protocols, we propose that the base station can download a suitable last hop protocol to the mobile at every point in time according to a specific policy.

The idea of downloading protocol code on the fly has two advantages: the mobile station doesn't need to cache the already received instances of protocol code (thus not wasting memory) and the base station doesn't need to keep track of the protocols available to the mobile. It can be a serious problem for the base station to keep the list of protocols in a correct state, especially if memory on the mobile is a scarce resource. This scheme can have the disadvantage of non-negligible overhead for transferring protocol code. However, we believe the code can be kept sufficiently small if some software infrastructure (*supporting functions*) is already available on the mobile station, e.g. timer support. An alternative to this idea is to use a single protocol which is capable to achieve a wide range of behaviour by dynamically adjusting appropriate protocol parameters (e.g. maximal packet size, number of retransmissions and so forth), but we think that it is very hard to build a single protocol capable to handle the whole range of wireless channel conditions or underlying bearer services or at least a large enough subset of it.

There are five basic questions which need to be resolved:

- We need a static mapping from different environmental circumstances (link quality, radio technology, ISP policy, ...) to last hop protocols.
- We need additional criteria for restricting protocol switching (e.g. in order to avoid

heavy oscillations)

- When to switch between protocols?
- How to transmit the protocol?
- How to switch between protocols?

In this paper we cover only the last three questions with main focus on the question "How to switch".

To be more specific, we want to attack the following problem: how can an abrupt change of the last hop protocol be achieved while preserving a reliable service with no losses and duplications, such that all existing TCP connections can be preserved? With "abrupt" we refer to the case that often no graceful teardown of the current last hop protocol is possible. We are mainly interested in the necessary protocol mechanisms in the mobile terminal and do not cover the problem of transferring TCP contexts from one base station to another. Due to this, it suffices for us to only look at the case of a single base station, which downloads a new last hop protocol from time to time. However, we assume that our results are valid for the case of multiple base stations too, if the necessary handovers of the TCP contexts can be carried out fast enough.

We assume, that the dynamical download of protocol code is initiated by the base station, because it has sufficient resources for storing protocol code and it has the necessary knowledge on network conditions.

3.1 Some Problems of Dynamic Protocol Switching

In this section we discuss the the main problems which need to be solved.

As possible candidates for LHPs we consider e.g. the classical ARQ-protocols (Send+Wait, Goback-N, Selective Repeat), hybrid protocols combining ARQ and FEC (Forward Error Correction) as discussed in [13], protocols with packet duplications [2], and so forth, but the class of LHPs is not restricted to these examples. Most protocol mechanisms can be simple, since they have to cover only one hop, e.g. there is no need for congestion control mechanisms.

We illustrate the problems of protocol switching with an example, namely the process of switching between Selective Repeat and Send+Wait. In Selective Repeat both sender and receiver maintain a *window*, i.e. a buffer with packets currently in transit, and also a sequence number. The Selective Repeat sender has in its buffer a subset of unacknowledged frames (which includes always the oldest frame) and another subset of acknowledged frames. The receiver has a subset of already received packets (where he has sent acknowledgements) and another subset of outstanding packets, to which the oldest buffer belongs. In contrast to that, in Send+Wait the buffer size of the sender is one and the receiver has no buffer at all, just a sequence number which helps distinguishing new frames from old ones. The switching algorithm now has to cope with the following problems:

- How are acknowledged frames other than the oldest one handled at the sender side?
- How are acknowledged but not delivered frames on the receiver side handled?
- How can base station and mobile achieve a common understanding on the time the protocols are actually switched in the presence of transmission errors on the medium?
- The meaning of sequence numbers and of acknowledgements (e.g. positive or negative acks) can vary between different ARQ protocols.
- How can different protocols be uniquely distinguished from each other?
- Which protocol is actually used for downloading protocol software to the mobile, especially if a mobile is newly switched on?
- How to guarantee that the sequence is maintained even when the LHP is changed?
- How to detect losses and duplicates even when the LHP (and more special: the maximum packet size) has changed?
- How should a generic LHP protocol interface be designed, if the environment should not need to know about the internals of a LHP?

In addition to that, the wireless channel does not vary only in time, but also in space, i.e. there can be completely different channel conditions from the base station to two different mobiles. To this behalf the base station must be able to operate many different protocols in parallel, for every mobile station one.

Often there are many mobiles within one wireless cell. We assume the mobile stations to be heterogenous with respect to computing power, processor types, operating systems, display capabilities and so forth. As a consequence the last hop protocols should preferrably be written in a system independent language, which allows for loading and executing code on the fly, like e.g. Java[1] or Caml Light [1]. The language used should in addition have a concept for dynamic libraries, since in order to keep the protocol code small, every mobile station

[1]Java is a trademark of SunSoft.

must provide some common basic services to the last hop protocols which are dynamically bound to the protocol code. Such basic services typically include access to timers, sending and receiving packets, data structures (e.g. lists) and so forth.

One important question to answer is to find out proper conditions for when to switch between protocols. The decision to switch to a new protocol should be taken by the base station in order to keep mobile complexity low. The base station has to evaluate some measures regarding link quality (e.g. the number of missing acknowledges, the received signal strength and so forth) and then to select a proper protocol. However, great care needs to be taken to avoid heavy oscillations. The selection algorithm should be able to distinguish between fast transient variations (e.g. fast fading due to movement) and slow variations in link quality (slow fading due to change from a rural to an urban environment) and, if possible, switch to new protocols only in the latter case. Clearly the objective is to keep the overhead for actually transferring the protocol code and for performing the protocol switching low while achieving good transmission performance. The concrete policy is a topic of further study.

In addition to the problems already mentioned, one has to consider the very important problems of authentication, i.e. who is allowed to load new protocol code into the mobiles, and security issues, i.e. how can the mobile station be protected against malicious or malfunctioning protocol code? These questions are easy to answer in a trusted environment, where the base station and the mobile belong to the same organization.

Chapter 4

An Architecture for Dynamic Switching of Last Hop Protocols

In this section we develop our architecture for implementation of dynamic switching of last hop protocols. An overview about the software- and protocol architecture of our approach is given in fig. 4.1.

The first observation from fig. 4.1 is that the last hop protocol shown in the remote socket architecture overview (fig. 2.2) is replaced by the DLLLP Client (*Dynamically Loadable Link Layer Protocol*) on the mobile station and the DLLLP Server on the base station, both gray shaded. In our approach, the DLLLP Client and Server have the following responsibilities:

- Providing the LHP service as defined in 2.2 to the local socket module and the export socket server, using an unreliable packet service. The service must not be affected by switching the last hop protocol.
- Secure and reliable transmission of protocol code (from base station to mobile)
- Mobile: receive packets with protocol code and perform switching between protocol instances.
- Base Station: evaluate link quality, select a new last hop protocol, retrieve the protocol code from the protocol database and transmit the protocol code to the mobile.
- The base station needs to detect, when the mobile has switched to a new protocol. Then the base station must also immediately switch to the new protocol.
- Determine link outages, i.e. longer disconnected periods, where no frame has been received from the peer station.

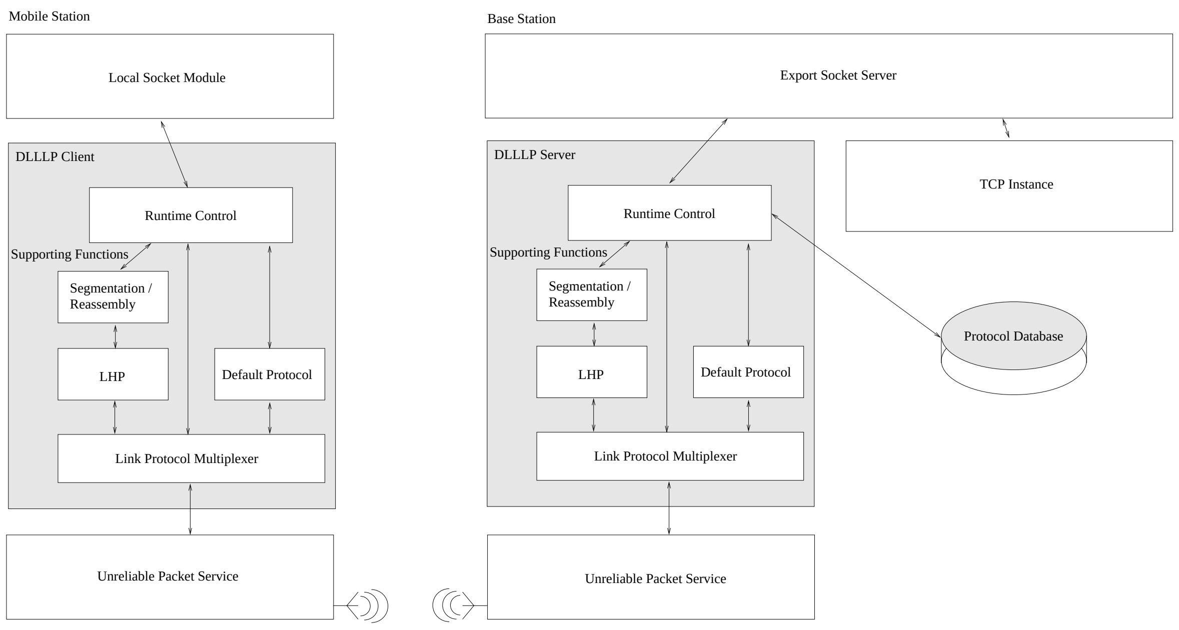

Figure 4.1: Software- and Protocol Architecture

- Provide a set of basic services to the (dynamically loaded) last hop protocols. This includes access to timer facilities, buffer access, access to the unreliable packet service, and so forth.

4.1 Definitions

Before we proceed with the architecture, we need to define some notions. Every station (mobile station, base station) associates one of two possible states to the wireless link: the *good* state or the *bad* state. Our intention is to distinguish between phases where communication is possible and phases, where communication is very error prone or impossible. We define a *link outage* event to occur exactly at the crossing from good state to bad state. A *living period* starts with the link changing from bad state to good state and ends with the next change from good state to bad state. With every living period we associate uniquely an *incarnation-ID* (usually increasing integer numbers) and an *LHP context*, explained below. Please be aware that during every living period several last hop protocols can be operated.

4.2 Link Protocol Multiplexer and Default Protocol

Between the base station and every single mobile station at every point in time two protocols are handled: the first one is the last hop protocol, the second one we have called the *default protocol*, see fig. 4.1. The default protocol is a fixed and robust protocol providing a reliable service. It need not to deliver good performance but it is more important to allow for data transmission even over bad links. It is mainly used for transfer of memory blocks of variable size (aka protocol code) and is always available. In addition it can be used for transfer of statistical data from the mobile to the base station (which can be used for determining the link quality), thus it has multiplexing capabilities. But it is not used for user data transfer. The concrete choice of a default protocol is a subject of further study.

In order to distinguish between the different protocols (the default protocol and the current last hop protocol) we introduce the *link-protocol-multiplexer*, which offers basically an unreliable datagram service and adds to every of its service data units an appropriate *protocol-ID*. To every distinct protocol (default protocol and the current last hop protocol) we associate uniquely an own protocol-ID. The protocol-ID of the default protocol is considered to be “well-known”. In addition to the protocol-ID the link protocol may add other framing functions, e.g. checksums. Every mobile station uses only the protocol-ID for multiplexing, while the base station additionally uses the mobiles address in order to support multiple mobiles.

4.3 Keepalive Protocol

We introduce a keepalive protocol between the base station and the mobile. We introduce the protocol for two different purposes:

- It provides a convenient method for determining the good and bad link state as follows: the link is associated the good state, if the time difference between the current time and the time instant where the last frame was received, is below a given threshold T_{TO}, and it is considered to be in the bad state otherwise.

- Since the base station maintains the TCP contexts it should be able to close the TCP connections (using RESET packets) in the case that there is no connection to the mobile and there is no other base station to which the TCP contexts can be moved (handover). Thus we need a kind of *soft state*.

The keepalive protocol works as follows: the link protocol multiplexer instance in the base station generates during idle periods for every mobile in fixed intervals of length $\frac{T_{TO}}{n}, n \in \mathbb{N}, n > 1$ short ping packets, which are answered immediately by the link protocol multiplexer instance of the mobile station. Under this protocol a bad channel condition occurs only, if n subsequent ping packets (or answers) are lost and no other packets are received from the peer. In this case the link protocol multiplexer generates a special signal (*link-outage signal*) for the upper layers, more specifically, the LHP, the runtime control and the export socket server or local socket module are informed. The design and parametrization of the keepalive protocol should take power saving considerations into account.

4.4 Resynchronization after Link Outages

We define, that after occurence of a link outage event all currently existing TCP sockets on the detecting station are released (as a result, the TCP connections are reset). In order to keep both sides synchronized, the peer station must be informed about this at the next time when communication is again possible. For this reason we define the following protocol: both stations (mobile and base station) maintain a local copy of the current incarnation-ID. As already mentioned, this incarnation-ID is maintained to be unique for every living period. A station copies its local incarnation-ID into every packet it transmits (this is performed within the link protocol multiplexer). At the very beginning the mobile registers itself at the base station and the base station assigns and transmits an initial incarnation-ID to the mobile (which stores it in a local copy). If a station gets a link outage event, it releases all of its TCP sockets and increases its local copy of the incarnation-ID. All future packets of the station carry the new incarnation-ID. If otherwise a station receives a packet with an incarnation-ID newer than its local copy, it releases all its TCP sockets and assigns its local copy the newly received value.

4.5 Protocol Switching

In order to explain the switching algorithm we first describe how data units are processed within our protocol architecture.

On the sender side Socket calls are transported first to the segmentation and reassembly layer (SR layer), which has knowledge about the maximum MTU size of the current last hop protocol. The segments are called LHP-SDUs. The SR layer maps the sequence of socket calls of all currently open sockets into a common single byte stream, every byte associated

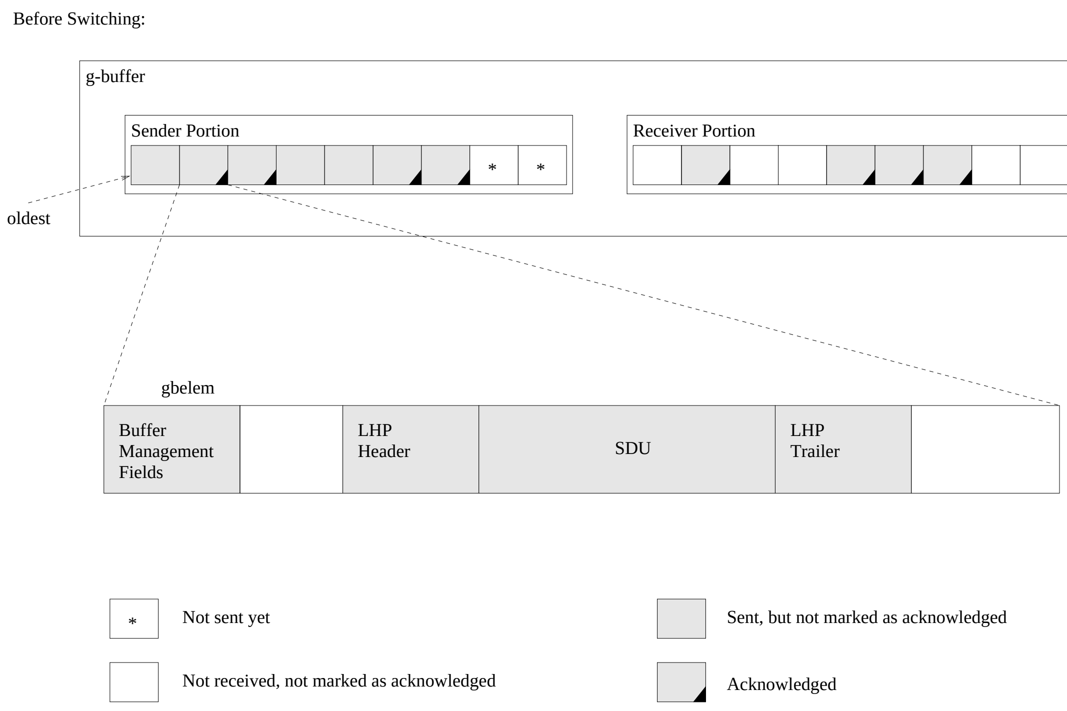

Figure 4.2: Structure of g-buffer and gbelems before switching

a byte count (position within the byte stream). Additionally special markers for the last byte of a socket call are introduced, in order to mark boundaries of socket calls within the byte stream. The SR layer fragments the byte stream into LHP-SDUs (according to the LHPs maximum MTU size), each fragment carries the byte count of its first byte and the end marker, if present (within a single LHP-SDU all bytes belong to at most one socket call). These LHP-SDUs are stored in a buffer, which we call *g-buffer* and which consists of a set of g-buffer-elements, each large enough for carrying a single LHP-SDU and information of the LHP-PDU header and trailer, such that these can be constructed in-place by the LHP. Such a g-buffer-element is denoted as *gbelem*. The structure of the g-buffer and the gbelems before and after switching are shown in figs. 4.2 and 4.3.

The g-buffer belongs to the so called *LHP context*. The LHP context is basically a storage for nontransient data, with their lifetimes equal to a living period, which is in general larger than the lifetime of a single LHP. This g-buffer must be used by the LHP as its sending buffer and receiving buffer, e.g. containing the window of Goback-N. However, we require, that the LHP can actually build LHP-PDUs within the g-buffer (gbelems), but it must not physically free gbelems, even if the LHP considers them to be transmitted successfully. Instead, in order

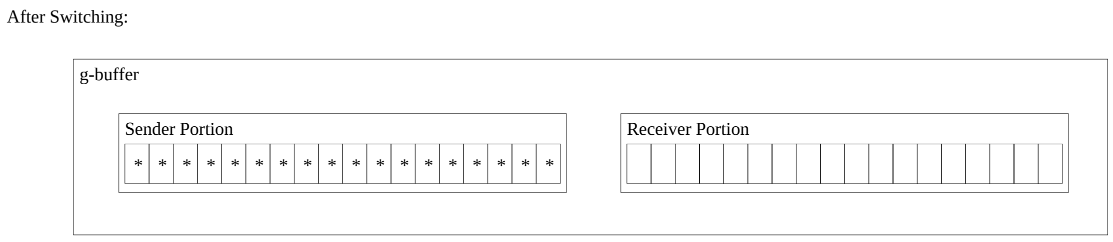

Figure 4.3: Structure of g-buffer and gbelems after switching

to signal proper transmission of a gbelem, the LHP must call a special g-buffer management routine, which only marks the gbelem as successfully transmitted. The gbelems are only freed by this management routine, if the oldest gbelem is acknowledged. The reason for this is to prevent losses of frames when switching. In addition to the raw gbelem entries the LHP may have packet related data, which have only local significance to the LHP, e.g. timers, sequence numbers, acknowledgement information and so forth. However, these local data are not stored within the LHP context.

Within a station at the time instance of protocol switching the following happens: first the old LHP is stopped, i.e. some teardown routine is called, which essentially resets its local data (resetting timers, freeing memory and so forth) without changing the state of the g-buffer and without processing further frames to send or receive (including acknowledgements). In the next step, all marks for successful transmission of gbelems are removed from the g-buffer (by the environment, not by the LHP), thus all gbelems are considered to have not being sent yet. For the receive buffer portion of the g-buffer all gbelems are freed[1]. If the MTU size of the LHP has changed, the g-buffer must be re-segmented by the SR-layer, i.e. the SR layer discards the old segmentation information, virtually restores the byte stream and performs a new segmentation corresponding to the new MTU size. After that, the new LHP is initialized and it starts transmitting all the elements of the new g-buffer according to its protocol rules. As a result it may happen that some fragments of the byte stream are transmitted twice[2], however it is assured, that no data is lost (modulo link outages). Duplications now can be detected by the receiving SR layer, using the byte counts. As a result, the service user of the

[1]We assume, that all gbelems of the receiver portion, which are received in sequence, are removed immediately from the g-buffer, thus at any time the oldest gbelem of the receiver portion is not yet received.

[2]Consider e.g. switching from Selective Reject to Send and Wait, where the MTU size remains constant. The g-buffer may carry the gbelems 1 to 7, with element 1 unacknowledged, elements 2 and 3 acknowledged, the rest unacknowledged. After switching all elements from 1 to 7 are considered unacknowledged, thus elements 2 and 3 are transmitted twice.

SR layer receives an in-sequence, error free stream of socket calls without duplications.

The method of simply discarding all out-of-sequence but correctly transmitted gbelems is justified by the observation that the propagation delay on the wireless link is typically very short (for a distance of 200 m and with speed of light equal to 200000 km/sec the propagation delay is 1 μsec) and thus, for most ARQ protocols which work with buffers on the sender side (e.g. Goback-N and Selective Repeat), a small buffer size suffices. As a result, the number of packets transmitted multiple times will be small.

After the old last hop protocol was properly stopped as described above, the new protocol must be started. It needs to be instanciated and initialization routines need to be performed. An integral part of the initialization is the binding of the software environment to the protocol instance, i.e. the protocol instance then has access to the basic *supporting functions*, provided by the DLLLP Client in case of a mobile or the DLLLP Server in case of the base station. These supporting functions may include memory management, timer management, some common data structures, debug facilities and access to statistical counters (a good starting point for necessary functions would be [10]).

The next issue to resolve is actually when to start the process of switching between protocols. The mobile performs switching in the same moment, where it has correctly received a whole block of protocol code (via the default protocol) and has successfully performed some consistency checks (e.g. check whether the received object has some needed functions via object introspection). After the conformance test the mobile starts the switching procedure as described above. This procedure is assumed to be "atomic" w.r.t. packet transmission and reception of a new LHP. On the base station we need another switching scheme, since the BS cannot estimate the exact switching time of the mobile (we assume, that no protocol instance can handle protocol data units of other protocols, thus switching is really necessary). To this behalf we associate uniqely to every distinct last hop protocol a *protocol-ID*. This protocol-ID is transmitted in the header of every frame (in fact, it is added by the link-protocol-multiplexer). Using this protocol-ID the base station can determine from frame to frame whether the mobile has actually switched the protocol.

4.6 LHP Interface

In order to work properly within the environment described so far, every LHP must conform to a special *LHP interface*, which describes the relationships of the LHP to the upper and lower protocol instances. From the lower protocol layer the LHP expects an unreliable datagram service, which can lose, corrupt, duplicate or reorder packets. From the lower layer it is assumed that it can calculate checksums, if required so by the LHP (on a per-packet basis). Basically the lower layer only provides two service primitives: request and indication, there is no confirmation. The LHP itself is responsible for implementing a reliable service with error-free in-sequence delivery without duplications during the lifetime of the LHP. It must support request and indication primitives and it must be able to create link error primitives (see section 2.2), in addition to that it signals only link disconnects to the upper layers. Its public interface must contain provisions for protocol teardown, protocol instantiation (including the above mentioned binding to the supporting functions), status and statistics reports (e.g. number of retransmissions), methods for notifying the arrival of new LHP-SDUs from the service user, and a method which is called on the link-outage-event (discarding all gbelems and re-initialization).

4.7 Runtime Control

The DLLLP Client and Server both contain a module for *runtime control*. On the client side the runtime control is mainly responsible for controlling the switching process and for gathering some statistical data concerning transmission results (e.g. number of lost frames, number of retransmissions). These data are sent periodically to the base station (via default protocol), which can use them in estimating the link quality. The runtime control on the server side has some more responsibilities. It must constantly monitor the link quality, taking measures as signal strength or number of lost and retransmitted frames into account. This is performed for every single mobile, since in wireless environments the channels between different stations separated in space are often in different states. Based on this and on the time of last LHP switching a proper LHP is selected. If the chosen LHP differs from the one currently in use, the new protocol is retrieved from the *protocol database* and downloaded to the mobile using the default protocol.

Chapter 5

Conclusions

In this paper we have motivated, why an active network approach, which allows the dynamic downloading of protocol code within the remote socket architecture is helpful for efficient wireless internet access and global mobility. After introducing the remote socket architecture we have taken two steps: in the first step we identified the key issues which need to be resolved for abrupt switching of protocols while maintaining TCP connectivity, in the second step we developed a protocol architecture and suitable algorithms for downloading protocol code and protocol switching on the fly.

Currently we are developing a first proof-of-concept implementation of our architecture, which focuses mainly on the switching mechanism. To this behalf, we have chosen to implement all the protocol entities described so far within user space in Linux, using Java. As the underlying unreliable packet service we use UDP. This will provide a suitable testbed for our switching algorithm. However, in order to investigate the performance gain of our architecture, a kernel space implementation needs to be developed. For this purpose we evaluate different possibilities for running machine independent code within the Linux kernel (e.g. porting a Java Virtual Machine or implementing a p-code interpreter).

There are several attractive topics for further research. We have already mentioned the choice of an appropriate policy for protocol switching and also an appropriate default protocol. Another interesting issue is the possibility to introduce a *packet classifier* within the runtime control, which allows for prioritization of certain classes of TCP connections by inspecting e.g. the port addresses. One application example is to handle telnet sessions with higher priority than ftp sessions. Again, it is worthwhile to think on making this prioritization scheme also downloadable.

Bibliography

[1] D. Scott Alexander, Marianne Shaw, Scott M. Nettles, and Jonathan M. Smith. Active Bridging. In *Proc. of ACM SIGCOMM'97 Conference,*, Cannes, France, September 1997.

[2] A. Annamalai and Vijay K. Bhargava. Analysis and Optimization of Adaptive Multicopy Transmission ARQ Protocols for Time-Varying Channels. *IEEE Transactions on Communications*, 46(10):1356–1368, 1998.

[3] Robert Arnott, Seshaiah Ponnekanti, Carl Tayler, and Heinz Chaloupka. Advanced Base Station Technology. *IEEE Communications Magazine*, (2), February 1998.

[4] H Balakrishnan, V. Padmanabhan, S. Seshan, and R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, 5(6):756ff, 1997.

[5] Eric A. Brewer, Randy H. Katz, Yatin Chawathe, Steven D. Gribble, Todd Hodes, Giao Nguyen, Mark Stemm, Tom Henderson, Elan Amir, Hari Balakrishnan, Armando Fox, Venkata N. Padmanabhan, and Srinivasan Seshan. A Network Architecture for Heterogeneous Mobile Computing. *IEEE Personal Communications*, 5(5):8–24, 1998.

[6] M. Bronzel, D. Hunold, G. Fettweis, T. Konschak, T. Dölle, V. Brankovic, H. Alikhani, J.-P. Ebert, A. Festag, F. Fitzek, and A. Wolisz. Integrated Broadband Mobile System (IBMS) featuring Wireless ATM. In *Proc. of ACTS Mobile Communication Summit '97*, Aalborg, Denmark, October 1997.

[7] D. D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *Computer Communication Review*, 20(4):200–208, 1990.

[8] A. DeSimone, M. Choo Chuah, and O. Yu. Throughput performancce of transport-layer protocols over wireless lans. *Proceedings IEEE GLOBECOM '93*, pages 542–549, 1993. Houston.

[9] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric A. Brewer. Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives. *IEEE Personal Communications*, 5(4):10–19, 1998.

[10] Norman C. Hutchinson and Larry L. Peterson. The x-Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17:64–76, January 1991.

[11] Randy H. Katz, Eric A. Brewer, Elan Amir, Hari Balakrishnan, Armando Fox, Steve Gribble, Todd Hodes, Daniel Jiang, Giao Thanh Nguyen, Venkata N. Padmanabhan, and Mark Stemm. The Bay Area Research Wireless Access Network (BARWAN). 1996.

[12] M. Kojo, K. Raatikainen, M. Liljeberg, J. Kiiskien, and T. Alanko. An efficient transport service for slow wireless telephone links. *Selected Areas in Communications*, 15(7), 1997.

[13] Hang Liu, Hairuo Ma, Magda El Zarki, and Sanjay Gupta. Error control schemes for networks: An overview. *MONET – Mobile Networks and Applications*, 2(2):167–182, 1997.

[14] Berthold Rathke, Morten Schläger, and Adam Wolisz. Systematic Measurement of TCP Performance over Wireless LANs. TKN Technical Reports Series TKN-01BR98, Technical University Berlin, 1998.

[15] Morten Schläger, Berthold Rathke, Stefan Bodenstein, and Adam Wolisz. Advocating a Remote Socket Architecture for Internet Access using Wireless LANs. *MONET - Mobile Networks and Applications*, 1999. to appear.

[16] W. R. Stevens. *UNIX Network Programming.* Prentice Hall, 1990.

[17] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, jan 1997.

[18] Helen H. Wang, Randy H. Katz, and Jochen Giese. Policy-Enabled Handoffs Across Heterogeneous Wireless Networks. 1999.

[19] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated Volume 2 - The Implementation.* Addison-Wesley, Reading, Massachusetts, 1995.