A Programmable Service Architecture for Mobile Medical Care

Rajiv Chakravorty, Suman Banerjee, Ian Pratt

ABSTRACT

We introduce Mobicare - a flexible, programmable architecture that efficiently exploits mobile and wireless communication systems to provide better healthcare services in a wide-range of scenarios. The Mobicare architecture consists of three important building blocks: a body sensor network (BSN) consisting of wearable sensors and actuators with wireless inter-connections; a BSN Manager (also called Mobicare client) that connects the BSN to an 'always-on' wide-area communication interface using GPRS or UMTS cellular wireless links; and back-end infrastructure support (Mobicare servers) at healthcare providers to implement necessary healthcare functionalities. A novelty in Mobicare is the remote dynamic software update functionality applied to the native code of the client device. We define the mechanisms for registration and remote configuration of the body sensors, as well as remote health data services such as health information downloads and diagnosis data uploads with the provider servers. We implement a prototype for Mobicare as a proof-of-concept, and evaluate it in an experimental wireless testbed consisting of Bluetooth and GPRS/UMTS cellular networks. Our evaluation demonstrates Mobicare as a feasible and useful infrastructure paradigm for next generation healthcare.

1. INTRODUCTION

A significant proportion of the human population suffer from various medical conditions, including chronic ailments and medical emergencies due to sudden injuries. In absence of continuous medical care, many chronic ailments prove to be fatal. On the other hand in various medical emergency scenarios, timeliness of medical attention is even more important. In many such cases, e.g., cardiac arrest, the risk to a patient's life can be considerably minimized by improving the quality and timeliness of medical care in the "golden time window" immediately following the injury. However under the existing healthcare systems, the fatality rate in the US from heart failures itself is more than 42%, many of which are due to delays incurred in initiating medical intervention.

In this work we propose MobiCare — a flexible, programmable architecture that enhances mobile healthcare services by enabling such real-time, continuous, and timely monitoring of patients thereby enhancing quality of care for patients and potentially saving many lives. Some important benefits of a system like MobiCare include:

Continuous Monitoring for Chronically-ill patients: Remote monitoring enables chronically-ill patients to conform to long-term course of medical treatment that can considerably reduce the crisis and relapse rate for such patients.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

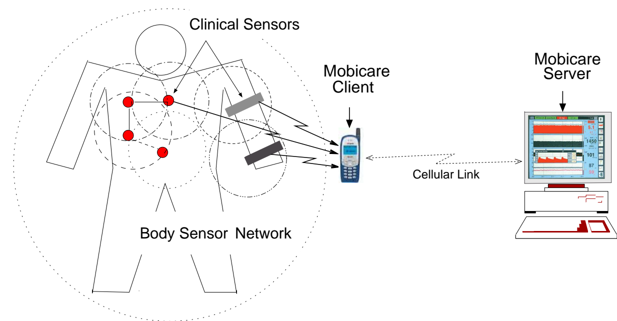

Figure 1: Schematic for Mobicare.

Better Quality Care and Feedback: By enabling more effective monitoring of patient's condition, MobiCare provides more accurate and useful information to medical personnel which ultimately leads to better medical advice and feedback to patients. This will lead to better treatment of ailments and an overall improvement in the quality of care for patients.

Increased Medical Capacity: Medical centers using MobiCare can treat many more patients. Hospitals often have patients with ailments which require a long recovery period. Using MobiCare many such such patients can be very effectively monitored and treated in their homes. This offers the potential for increased medical capacity and personalized healthcare.

Reduced Medical Cost: The proposed mobile healthcare mechanisms reduce relapse rates of ailments and hospitalization period for patients. It also reduces the need for frequent medical consultation. By ensuring such reductions, the mobile healthcare system can significantly reduce medical costs.

Healthcare systems today have failed to efficiently exploit continued advances in mobile and wireless systems, despite their significant advantages, to provide better healthcare services. Our goal in this work has been to effectively exploit such recent advances to define a more efficient healthcare infrastructure. MobiCare consists of three important building blocks: a body sensor network (BSN) consisting of wearable sensors and actuators that are inter-connected using the wireless medium; a BSN Manager (also called the MobiCare client device) that connects the BSN to an 'always-on' communication wide-area interface, e.g., a GPRS/UMTS cellular link; and backend infrastructure support (servers) at healthcare providers that provide necessary healthcare services to patients. Mobicare is a 'programmable' architecture where client devices can be dynamically updated with new medical system softwares, features and applications. Mobicare also defines the mechanisms for remote registration and configuration of the body sensors. Together the system provides various health data services such as health information downloads and diagnostic data uploads between

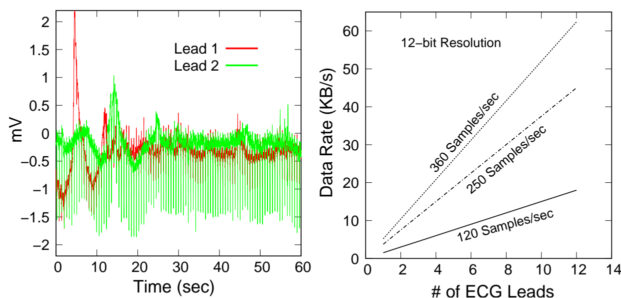

Figure 2: Plot shows (a) capture of two ECG signals for a patient with congestive heart failure, and, (b) data-rate requirements when number of the ECG leads are varied. The ECG signals were sampled at 250 samples per-second with 12-bit resolution over a range of 10 millivolts. (ECG data source: http://www.physionet.org)

patients and healthcare providers.

Enablers of MobiCare

A number of recent innovations enables the efficient design of the MobiCare architecture. Advances in medical sensors today enable efficient, remote monitoring of patients. For example, sensors to measure ECG are now commercially available from Numed [6] and Health Frontier [3]. Other sensors from Nonin [5] and Linde AG [4] use wireless connectivity (Bluetooth-based) to provide remote monitoring of vital body signs. The CodeBlue project at Harvard has also developed (using the Berkeley MICA2 mote) a low-power, low-frequency, wireless pulse oximetry and ECG sensor for patients [14]. With clinical sensing technology advancing at a much faster rate one can expect a range of such energy-efficient wireless medical sensors devices to become available. We will exploit such sensors to construct the BSN that monitor patient health non-invasively to gather vital health data, e.g., heart condition, blood pressure, serum glucose level, temperature, oxygen saturation (0_2). The sensors in BSN use a wireless interface to communicate such data to the BSN Manager and ultimately to the back-end servers.

We also exploit the conitnued increase in coverage and bandwidth of cellular wireless networks worldwide to build the 'always-on' wide-area interface of the BSN. For instance, the newly deployed 3G systems provide for data-rates that are much higher than offered by conventional fixed dial-up modems. Such networks therefore open up the possibility for patients to be continuously monitored and their vital health data to be very efficiently transported from the BSN to back-end servers, thereby enhancing the timeliness and quality of medical care.

Design Goals of Mobile Health System

In order to identify the key design goals for a mobile health care system, we consider the monitoring requirements for patients. Figure 2 shows an example of monitored ECG data from a heart patient and the corresponding data-rate requirements when the number of such ECG leads are varied. Figure 3 summarizes the typical time requirements and relative priority of some vital body signs including blood pressure, blood gases (O_2 and CO_2), heart condition (ECG), and

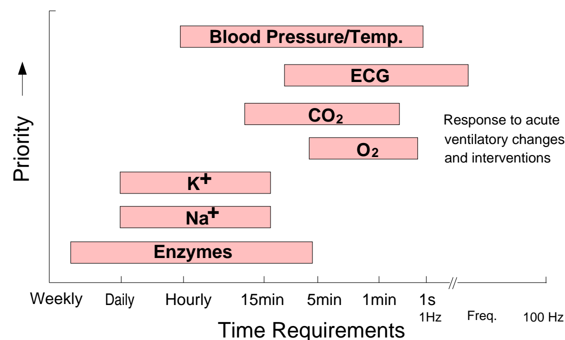

Figure 3: Typical priority and time requirements in clinical measurements.

enzymes. Thus design of a mobile healthcare system should consider the medical requirements to derive the design goals. We make the following observations.

First, vital body signs have different *time requirements* in patient monitoring (Figure 3). Such monitoring may be needed periodically or on-demand in real-time. Time requirements of such monitoring vary significantly – from few minutes to several hours – depending on the condition of the patient and the severity of the ailment. Flexible and remote configuration of sensors is therefore crucial for effective mobile medical monitoring. Second, clinical sensors that address different monitoring requirements of patients can potentially be built by different vendors and may use different wireless protocols and technology, each operating in a different part of the wireless spectrum. Such uncoordinated clinical sensor design (as exists today) can make integration and self-organization of such body sensors exceedingly difficult to achieve. In order for such sensors to function together in a single BSN, remote adaptation and reconfiguration should be an integral requirement of the architecture. Finally, due to its critical and real-time nature, reliability, security and timeliness of data delivery from the BSN to the back-end servers is crucial.

In the following we discuss how we accomplish these design goals in Mobicare:

A Programmable Architecture. A mobile health care system should be able to dynamically integrate, organize and configure new body sensors based on the needs and the requirements of patients and health providers. Mobicare fulfills this goal with a programmable architecture – by applying dynamic software update functionality to the native code of the client device. This feature has many benefits to offer: (i) **Customization** – New heterogeneous clinical sensors can be added and dynamically configured and customized to the monitoring needs of the patient, (ii) **Control** – It provides the necessary control to a health provider to configure and control the operation of the different sensors in a body sensor network, (iii) **Updates** – This enables new features or applications to be incorporated at run-time (e.g. a new MAC protocol for a sensor) that improves the quality and reliability of the device.

Flexible Service Components. Service requirements in clinical monitoring imposes additional timeliness and prior-

ity constraints on the monitoring system. For example, some vital sign data have higher priority than others. Mobicare enables service components to dynamically self-activate, (re)-configure, update, and customize so as to suit the monitoring needs of the patient and the health providers. By using these mechanisms, the service components are able to effectively address the *time* and *priority* requirements of health monitoring of patients.

Reliable, Secure and Time-bound Data Delivery. The nature of health data available from a patient sensor network requires reliable, secure and time-bound data delivery to the provider servers. However, data delivery over wireless cellular links can be challenging. Such links are plagued by problems of high and variable round trip times (RTTs) and relatively low bandwidths. Links occasionally experience 'stalls' due to the loss of coverage (severe fades in the 'holes') and during handovers (device or patient mobility). Collectively, these issues exacerbate the challenges of reliable and time-bound data delivery over wireless cellular links. Mobicare overcomes these challenges through design of service protocols that helps to quickly adapt to the changing conditions of the underlying network. Additionally, Mobicare protocol design considers the 'nature' of the clinical data available and can effectively prioritize transmission of the health-critical data as required.

Contributions

We make the following important contributions:

- We propose the first architecture that efficiently exploits mobile and wireless communications systems for medical healthcare services.
- We introduce programmable service architecture as an integral component of any mobile healthcare system. This feature allows dynamic integration, configuration as well as control of diverse clinical sensors into the healthcare system.
- We implement a prototype for the Mobicare client and services, and evaluate them in an experimental wireless testbed consisting of bluetooth and GPRS/UMTS cellular networks.

Roadmap

Our paper is laid out as follows. The next section details the Mobicare architecture while Section 3 elaborates on its services description. Section 4 discusses the Mobicare client and server-end design issues while Section 5 elaborates at length the design and implementation of dynamic software update functionality for Mobicare clients. Section 6 presents our evaluation while we discuss some related issues in Section 7. Section 8 covers related work and the last section concludes our paper.

2. THE ARCHITECTURE

The Mobicare architecture (figure 4) consists of three components: the body sensor network, a client connected to wide-area communication infrastructure and backend support at the healthcare service providers.

We discuss each component in detail:

Body Sensor Network – Client Interface. A body sensor network (BSN) is a wearable network of medical sensors and actuators that are interconnected using the wireless medium. A Body Sensor Network Manager (BSNM) interacts with the BSN to aggregate data gathered by the body sensors. This body sensor network manager functionality is implemented in a 'Mobicare client' device that can interact with the health provider servers to offer mobile healthcare services to the patients.

The Mobicare client (or the BSN Manager) functionality is, therefore, implemented in a device which has a wide-area wireless network interface. It periodically monitors various clinical sensors to aggregate vital body signs and uploads important health information to the provider servers using a secure wireless communication channel, e.g., a cellular link.

A wearable device such as the IBM wristwatch [12] is well-suited to serve as a Mobicare client in mobile medical settings for patients. Other similar portable devices can also be used to implement such functionality, e.g., Intel's Personal Server [23] or a cellular phone. Such devices are power-efficient, user-friendly, and they provide the necessary wide-area wireless connectivity to interact with the local environments. In particular, a cellular phone readily offers wide-area wireless connectivity and is potentially the best-suited device for such use. The Mobicare client consists of standard "built-in" functions to offer flexible customized services to a patient as needed by the healthcare provider. These functions optimize use of link bandwidth or transmission energy of the patient's health data that are being actively probed by the body sensors, as well as some other costs and optimizations involved.

A Mobicare client offers remote dynamic software update functionality applied to the native code of a device. With dynamic update functionality, a Mobicare client device can customize itself to new medical applications and services, and new clinical sensors can connect to this client and be configured into an existing body sensor network. By using flexible service components can help reconfigure service-related parameters based on the instructions available from the healthcare vendor. This feature is very useful to patients that need constant monitoring of their vital body signs. For example, to measure the pulse rate and oxygen (O_2) saturation for a patient, an oxi-meter sensor is attached to the patient's finger while ECG electrodes (probes) are entwined on their body vests to monitor and acquire necessary health data. Once configured these service parameters can be used to probe and configure body sensors periodically, as well as upload the sensor data collected with the provider servers.

Non-invasive body sensors can monitor patient vital body sign even when they are *on-the-move* and following their daily activities. Sensor data collected this way is stored and then transmitted to the provider servers over the cellular data network. This enables round-the-clock monitoring of the patient's health by the health provider. Physicians can access individual (patient) data and can provide feedback. Alternatively, the provider can reconfigure (program) these sensors remotely using flexible available service parameters.

Mobicare Communication Infrastructure. Mobicare offers the ease of mobility for patients with cellular wireless connectivity. A Mobicare clients acts as the central unit to serve or connect to a nearby communication gateway using a public 2.5G, 3G cellular network or even a WLAN network. High speed mobile connectivity is readily available in most cellular handsets, and network support to handle

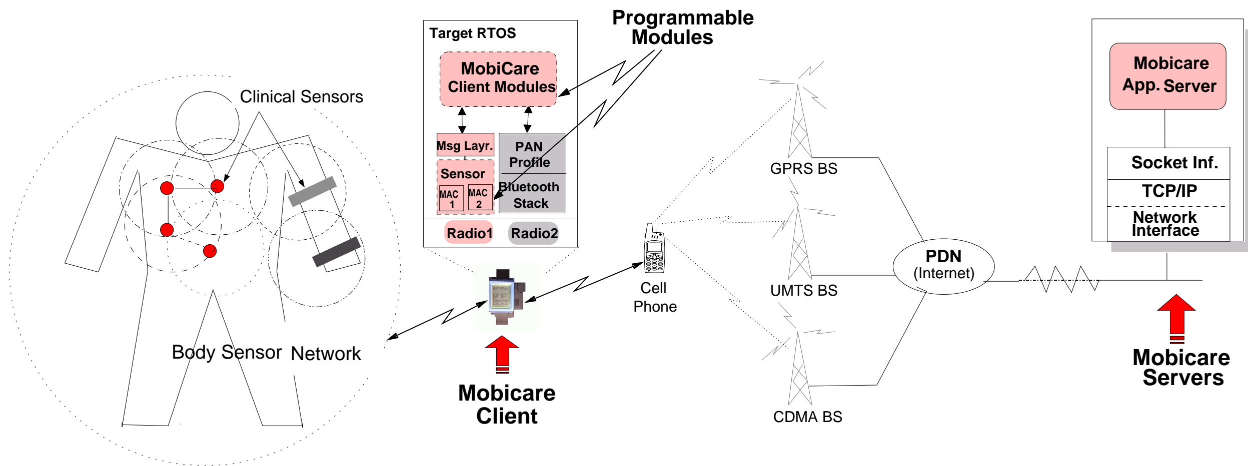

Figure 4: Mobicare System Architecture and Components. Dotted components of the client highlight modules that are dynamically reconfigurable (programmable). The 'Mobicare client' functionality is shown to operate in a wearable wristwatch device.

transmission and problems such as network disconnections (e.g. moving through tunnels) or network-supported adaptations (e.g. adaptation proxy) necessary during network handover events are also supported.

The 2.5G and 3G communication infrastructure helps transmit in real-time vital body sign data, video and images of patient activities via the cellular links to mobicare health providers. The improves the quality and timeliness of care provided to a patient and gives better information access to health providers for necessary action and feedback. During health emergency, the availability of vital signs (and video data if necessary) to a remote consulting physician can change the care provided during transport.

Healthcare Provider Support. Mobicare health providers provide back-end support for healthcare services for the purpose of trial and evaluation. Services offered by the center enables patients to be continuously monitored and specialists able to observe the evolution of patients, and intervene, if necessary. Mobicare health centers also provide continuous collection of biological data for patients e.g. ECG, temperature, blood glucose level etc.

Decision making by specialists located remotely forms an important component in Mobicare. Hence training strategies and decision aids are effective in supporting such remote decision making. These decisions aids also sometimes help manage the limitations in data made available from the medical systems and information support.

3. SERVICES DESCRIPTION

Mobicare defines the mechanisms for health care services as well as functions to activate and configure service-specific parameters. These functions include health information downloads and periodic uploads with the provider servers.

3.1 Protocol Definition

Mobicare uses of an application protocol built using standard HTTP. An application layer protocol enables health providers to reuse service infrastructures (servers), have easy access to existing services as well as the flexibility to compose new ones. Services in Mobicare are invoked using the standard HTTP protocol by submitting an HTTP request as a base URL (uniform resource locater) acting as a common access point for mobicare services. The name of the service is then appended to the base URL as the final path component, and arguments to each service are encoded and appended as URL query parameters. Consider the example:

```
http://www.mobicare.net/services/Activation?Select
```

In this URL **Activation** corresponds to the name of the service and **Select** gives the service step for Activation. To interact with the servers a Mobicare client makes use of the standard **HTTP POST** method in the request header along with the URL meant for that service. The benefit using the POST method is that it allows data to be sent to the server in a client request itself. This data is typically directed to a data handling program that server has access to (e.g., CGI, servlet). Unlike the HTTP GET method, the data sent to the server is in the body section of the client's request. After the server processes the POST request and headers, it passes the body to the server program specified by the URL.

Mobicare provides adequate parameter flexibility in its protocol definition. Any services can create custom parameters as suit their needs. The entity in the POST message constitutes service-specific parameters, and these services can pick amongst standard service parameters and execute the protocol service steps between the client and the server. The rules and syntax that governs Mobicare protocol are similar to those of standard HTTP.

3.2 Service Definition

Mobicare offers services such as the device activation, remote (sensor) configuration, health data services such as downloads and health diagnosis uploads as well as remote dynamic software update service for client devices. We discuss each of the service in detail.

3.2.1 Device Activation and Management

The device activation service in Mobicare enables client devices to self-activate and establish an account with the

health provider. This process is also known as remote registration (activation). While the device is typically activated once, it may go through multiple registrations as both involve similar protocol steps.

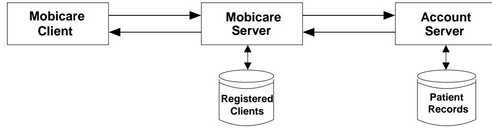

Figure 5: Device Activation in Mobicare.

For instance, a virgin client device when activated for Mobicare services initially will make use of a URL *preset* to connect to the Mobicare server (as shown in figure 5). The mobicare server makes use of a unique device identifier (ClientId) for a Mobicare client and logs corresponding device-specific data during the activation process. The account server uses this unique ClientId to create the patient record and returns an allocated account code in response to this initial connection data.

A Mobicare server associates the allocated account code corresponding to the ClientId of the device. This account code also serves to update other information such as the configuration of new and existing service parameters. The client device can also use an account code to query for different other services from the health provider and load Mobicare service-specific information available from the server. All service-specific information is stored in the persistent memory (e.g. flash) within the device for use later. In this way the client device logs and manages sensor-specific service information for its body sensor network.

3.2.2 Remote Configuration Service

The configuration service allow flexible composition and control over service parameters – new service parameters can be added or modified. These service parameters are present and stored in the persistent memory (flash) of the device.

Remote configuration service is very useful to health providers for remotely manipulating service-specific parameters within the client device. For example, this feature allows health providers to configure sensor service-specific parameters to manage and control settings of a body sensor network. Furthermore, it can also help fix a problem in the client device due to some misconfiguration of data and during remote dynamic device updates to help fix problems that require new settings for certain service-related parameters.

Figure 6 shows example steps during configuration service protocol exchange. In this example the server responds with the action `confirmConfig`, instructing the client to proceed with `confirmConfig` and to be followed by the writeConfig service steps. The `confirmConfig` step enables the server to check if there is a need to (re)configure service-specific parameters in the client. If the response of the server is a stop (*i.e.* configuration not required) or try later (*i.e.* server busy), the client will stop the sequence of actions. However, in the usual scenario the server responds with a number of service-specific identifiers to `read` their value. The client continues with the `writeConfig` message, presenting the identifiers and their values as requested. The response of the server in `writeConfig` holds the identifiers and their values. The client writes this new value back to the persistent

Figure 6: Configuration Service Protocol Exchange.

memory of the device.

3.2.3 Health Data Services

Mobicare data services are of two types: (i) download data services for health information downloads, and, (ii) upload data services for health diagnostic uploads.

Download data service in Mobicare is used for health information downloads to the Mobicare clients. This service is particularly useful for chronically-ill patients that need regular feedback about their health and vital body signs. However, other than the health information, download data samples may also include software modules (e.g. updates or upgrades), new applications (e.g. micro-browser) or even content (e.g. video or jpeg images) for the device.

Health diagnostic upload service allow client devices to collect and upload patient health data to the health providers. The health data is collected by periodically monitoring the body sensors and uploading this data to the health providers. The data is analyzed by the health providers to provide any health feedback (using the download service) if required.

Both upload and download service involve three-step protocol action. Both services initiate by sending the standard `queryService` message to the Mobicare server. The server responds with `Continue` and action `Download` for download service (or `Upload` for upload services). The client issues the next request `confirmDownload` (or `confirmUpload`) as an indication to the server that it is ready for information download (or that it has health diagnosis data for upload). The server responds to the client with the appropriate URL for upload server (or download server) for data to upload (or download). The client in final step posts the `doUpload` (or `doDownload`) message with health diagnosis data appended to its body. The server responds with success once all data is successfully uploaded (or download).

3.2.4 Dynamic Software Update Service

Dynamic software updates service is the key service that makes mobicare architecture ‘programmable’. A remote dynamic update service is useful in mobile heathcare settings for: (i) **Updates:** These are enhancements that improve the quality or reliability of the device. This can be a release for an existing piece of system software for the device (e.g. a

new MAC protocol for a new sensor) or a new medical application for the device, (ii) **Upgrades:** These are extensions that transform an existing client device into a new device that offer novel functionalities. (iii) **Application data:** This consists of application data (an application package) or content that is downloaded by client devices from their health providers. (more description in Section 5).

4. THE DESIGN

In this section we discuss in detail the design of the client and the server-end system in Mobicare.

4.1 Mobicare Client Design

A Mobicare client runs in an embedded device such a wearable wristwatch device [12], personal server [23] or a mobile phone. Recall that the main job of the client device is to manage the body sensor network on instructions from the health provider, implement Mobicare services and provide a network interface to the cellular wireless network. To offer a programmable architecture in Mobicare requires client components be implemented in a modular fashion as shown in figure 7.

In the following we discuss each component in detail:

Persistent Data Module (PDM). The PDM manages, stores and provides access to the service specific parameters in persistent memory (flash) of the client device. A virgin client device when activated retrieves this service-specific information from the server and stores them in the PDM. The PDM exports an API (or a secure API as an option) to be accessed by other client modules while communicating with the mobicare server.

Service Scheduling Block. Service scheduling block schedules service calls. After accepting service calls from other client modules it schedules them based on *first-come first-serve* basis. Alternatively, service scheduling with priority is also implemented. Scheduling services with priority is useful since in many cases uploads of critical health data should take precedence to less critical health data. During the device *first-time* boot-up process, appropriate calls are scheduled in this block for applications to boot-up. The block also maintains a separate timer to query the server (using `queryService`) periodically and may reschedule service calls postponed by the servers.

Communication Handler (CH). The communication handler implements the application protocol as service steps between mobicare clients and server. In this module, Mobicare service specific messages are assembled and sent to the server using the socket API supported by the target platform. As discussed earlier, the technique used here is the HTTP **POST** method for interacting with the Mobicare servers. A secure channel to the server may be requested (if available) as an option. The handler runs an independent task and communicates with other modules (Request Scheduling, Download and PDM) through target OS-specific IPC (inter-process communication) mechanisms. For certain service actions like activation (registration) and configuration, it can directly write the service-specific and user subscription related data to the PDM. However, during downloads (e.g. update, upgrades etc.), it invokes the functions of the download module that takes care of the downloaded data. It typically operates on a *per-call* basis. After receiving a response from the server, it performs message parsing for Mobicare specific parameters and if necessary schedules the next call to be made as defined by the protocol action for that service.

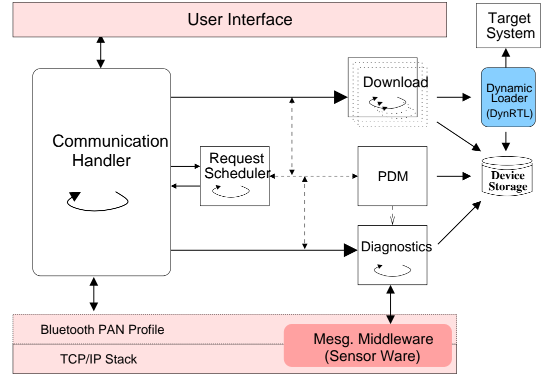

Figure 7: Mobicare Client-Side Design.

Download Module. The download modules unpacks modules available from the server, stores them in the device file-system and causes the dynamic loader to integrate the software module(s) in the client's software stack. This module is dynamically instantiated on availability of downloaded offers from the communication handler (available from the health server). Once the downloaded data is transferred to this module, it reads the headers of the download package (discussed in Section 5.3) to determine the type of downloaded data. *Note that the downloaded data can be an entirely new medical application; or a new release of an existing service module within the client device.* The data downloaded is then exported to its appropriate location to be stored within the device. The module also makes use of the dynamic loader module (discussed in Section 5.2) for loading and unloading modules from the device memory.

Health Diagnosis Module. The health diagnosis module (DM) periodically uploads health-related patient data from the mobicare client device to the health servers. The DM collates vital health signals from the body sensors, samples and filters them to report the patient data to the health server in an appropriate human-readable format. Depending on the severity of the vital health data received, urgent report uploads are possible using composable service protocol actions.

4.2 Server-end Design

The server-end design is shown in figure 8. The decomposition of the server-end system results into three different layers – the service layer, the function (protocol) layer and the resource layer. At the service layer the server-end design consists of the implementation of device activation, configuration, and data services such as health diagnosis uploads as well as information downloads. It also implements tools that enable remote dynamic software update services. Services at the server-end system are implemented using standard (protocols) functions such as the `queryService`,

`upload`, `download` and `configuration` functions.

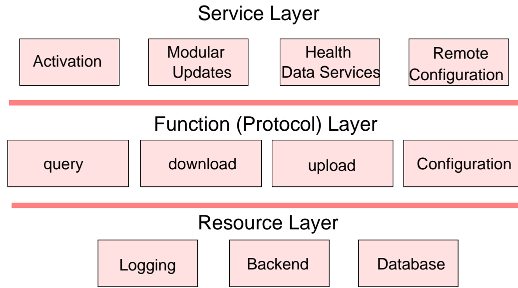

Figure 8: Server-end decomposition.

At the resource layer, mobicare server-end systems consists of the web servers with support for their own set of servlets (server programs), a SQL database, and one or more backend servers. Web servers run their own set of servlets (with JServ) and that also implements the service interface with Mobicare clients.

5. DYNAMIC SOFTWARE UPDATES

The novelty in Mobicare is the remote dynamic software updates functionality applied to the native code of the client device. As previously discussed, this functionality in devices helps to accelerate deployment of new mobile health services and applications in different ways. First, it allows for easy sensor device customisation – new diverse clinical sensors can be added to the body sensor network without consequence to other sensors. New sensors can be dynamically configured to join the body sensor network and this is very useful to the patients and providers. Second, it provides the necessary control to a health provider to configure and operate sensors in a body sensor network. Finally, software updates enable new medical features to be incorporated in the device at run-time (e.g. a new sensor MAC protocol) or even help fix software bugs. Such updates and upgrades can improve the quality and reliability of the client device.

In order to support dynamic software updates in mobicare clients, we need to adhere to an approach similar to that used in Microsoft's component object module (COM) model [22]. However, since most embedded real-time OSes lack this functionality, we consider in detail solutions that enable support of this functionality in an embedded client device.

Note that support for dynamic updates requires existing client modules to be able to offer that 'extra functionality' that requires it to be independent of other existing client modules. This additional functionality enables modules to be able to detach their interface and end instantiation when needed. Modules can also release other modules' interface when instructed to allow for its release.

This leads to the following requirements in Mobicare:

Modular code organization for embedded clients,

Wrapper tool Wrapping is the first step to prepare client modules for dynamic updates. The wrapper tool is a compile and link time tool that can proxy-patch modules and prepare them for dynamic binding in the client devices,

Dynamic Loader (DynRTL) This enables two important functions in Mobicare clients: (i) run-time dynamic binding including dynamic loading and unloading of modules, and, (ii) dynamic updates/replacements of modules.

PackBuild Tool A Server-end tool that packages modules in a format intended for easy distribution and download by the mobicare client devices.

In the following sections, we discuss in detail the design and implementation of the functionality that enables 'hot' modular updates in mobicare client devices.

5.1 Wrapping Modules for Dynamic Updates

Wrapping is the first step to enable dynamic modular updates in mobicare client devices. Mobicare offers a post-compilation tool called the **wrapper tool** that externally compiles and links individual client modules and prepares them for dynamic binding. The tool operates at various steps during the wrapping process of an object module before it is downloaded by the client device. The tools functions to read and interpret the ELF (Executable and Linking format) file format supported by our target client platform VxWorks. The tool reads multiple ELF files (the file to act upon and the file containing excluded symbols) along with their section headers, the symbol table, the associated string table, with the ability to modify the attributes of the symbols. The file layout remains unchanged by the wrapper tool – only the modified symbol table is written back to its original place.

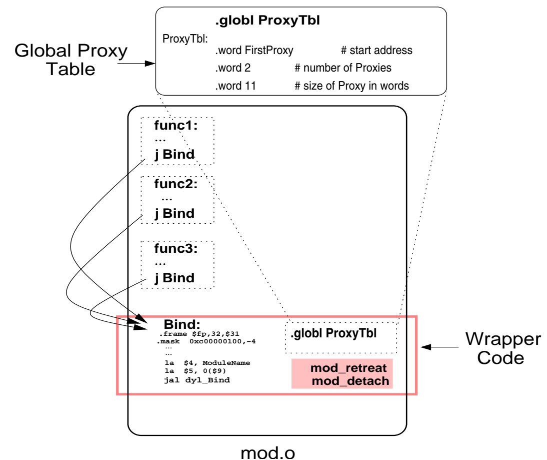

Figure 9: A sample wrapped module.

The chief function of this tool is to externally patch client modules with the "wrapper code" that prepares them for modular updates (figure 9). The tool operates in three steps. In the first step it reads the module and intercepts unresolved references to external functions and and inserts the proxy functions that invokes dynamic binding with the DynRTL loader (discussed in the next section) to resolve these function calls. The second step of the tool patches function code that enables existing client modules to detach (by providing the *Detach* function) from the (retreating) module being replaced. The final step makes use of the *retreat* function that announces a module that it is being replaced, asks

permissions for this replacement and performs the retreat actions.

The function <ModuleName>_Detach *resets* all proxy functions referring to a given target module. This module (actually its code segment) is delimited by begin and end addresses of the target OS. The following code fragment shows the *Detach* function, where this data structure is used:

```
extern "C" void  <Modulename>_Detach( void * start, void * end)
{
    uint * a = ProxyTbl.first;  // location of the first proxy

     // Reset proxy functions
    for (int i = 0 ; i < ProxyTbl.n ; i++) {
     uint * w = *a + ProxyTbl.addrOffset;
     if (*w >= (uint)start && *w < (uint)end)
        *w = (uint)(a + ProxyTbl.jumpOffset); //reset address
     a += ProxyTbl.size;
     }
}
```

Function Interception Code. To intercept external function references in client modules, the wrapper code makes use of the proxy functions. This interception code is applied to each individual modules by making use of an indirection (`jump`) instruction that redirects it to the `Bind` proxy function. The `Bind` function invokes dynamic binding (`dyn_bind`) with DynRTL to resolve the external reference. Figure 10 shows an interception code for function `afunc` and `Bind` proxy implemented using the MIPS instruction set.

The (MIPS) code for `Bind` is wrapped to the individual client modules using the wrapper tool. The `Bind` proxy function calls the function `dyl_bind` of the DynRTL module, inserts the address obtained into the proxy and re-executes this proxy function. The `dyl_bind` function resolves the function call and the procedure works as follows. First, it searches for the symbol in the system symbol table and if a symbol is found, returns with the memory address of that symbol. However, if the symbol is not found, it will search for all available modules (including the device file system) for that symbol. If the symbol is detected, it will load the corresponding module into the memory. It then returns the memory address of that function. Note that `dyl_bind` delays any request while a download operation is still in progress, until the download process is completed. This avoids the need to explicitly remove entries of any module from the system symbol table.

Dynamic Modular Replacements. Client modules implement additional functions that allow for dynamic modular updates. Notice that in figure 10 the `ModuleName` argument is passed to the `dyn_bind` DynRTL function. This is used by DynRTL for *client-server* administration of the resident modules in a client device. This administration of modules is needed by DynRTL when a module retreats and therefore must ask all other modules using it to detach.

The function <ModuleName>_Retreat takes care to detach *all* other client modules using this module. It delegates this task to `dyl_DetachFrom` of DynRTL, adding the `ModuleName` as an argument. The following code fragment shows the implementation:

```
extern "C" bool <ModuleName>_Retreat()
{
    if (!dyl_DetachFrom( &ModuleName))
       return false;
     return CanRetreat();
}
```

Also notice use of `can_retreat` in the <ModuleName>_Retreat function – this is useful for modules that are *active*. An active module runs its own task (a light-weight process), and is usually difficult to replace. This is because the internal state of a task is unknown.

One technique to use here could be to wait and first make sure that the module first gets into a state for it to be replaced. The idea here is to probe the internal state; Mobicare therefore makes use of a separate `can_retreat` function that are implemented by all other modules within the client device. Use of `can_retreat` enables a module to first deactivate itself before being removed. The `can_retreat` function: (i) investigates whether the internal state of the module allows a retreat, and, (ii) waits until ongoing function calls have returned, save pertinent status information in persistent memory to be picked up later by the module being replaced. However, a drawback using this technique is that it may introduce problems when replacing modules implemented by third party or legacy modules that may or may not implement a `can_retreat` function. A simple solution to this problem could be to externally patch such legacy modules with a dummy `can_retreat` function, although even for such cases the internal state of the module may still remain unknown.

5.2 Dynamic Loader (DynRTL)

A dynamic loader (DynRTL) in Mobicare offers run-time support for resolving inter-module function references (binding) as well as functions for dynamically loading and unloading of modules. DynRTL module makes extensive use of the target OS system symbol table. The symbol table contains externally accessible functions in the system and its associated memory addresses. DynRTL is responsible for its relocation, registration of entry points for existing and any new modules with the system symbol table and if needed its removal as well.

The DynRTL implements two important functions:

Resolving inter-module References. Requests usually originate from the 'proxy' functions located in the wrapper layer of the module. DynRTL helps to resolve a reference to functions available in the system. Upon reception of a request, first the system symbol table is consulted to check whether the function is already present in memory. If not, all available modules in a dedicated directory of the local file system are searched for the required function and - if found - the module containing the symbol is loaded into the memory. In both cases, the address of the requested function is returned. For the module or application itself, this mechanism is completely transparent.

Administration of the Modules. Whenever an inter-module function reference is resolved, a data structure in which inter-module client-server relations are recorded is updated. This data structure contains, for each module loaded in memory, a list of other modules that use it. When a loaded module must be unloaded, it must notify all clients of its forthcoming retreat. The `Retreat` function of the module to be unloaded delegates this task to the Dynamic Loader. In DynRTL, `Detach` functions of every known client module is called, which in turn resets all the proxy functions that refers to a particular retreating module. In this way, any module can be unloaded without leaving any dangling

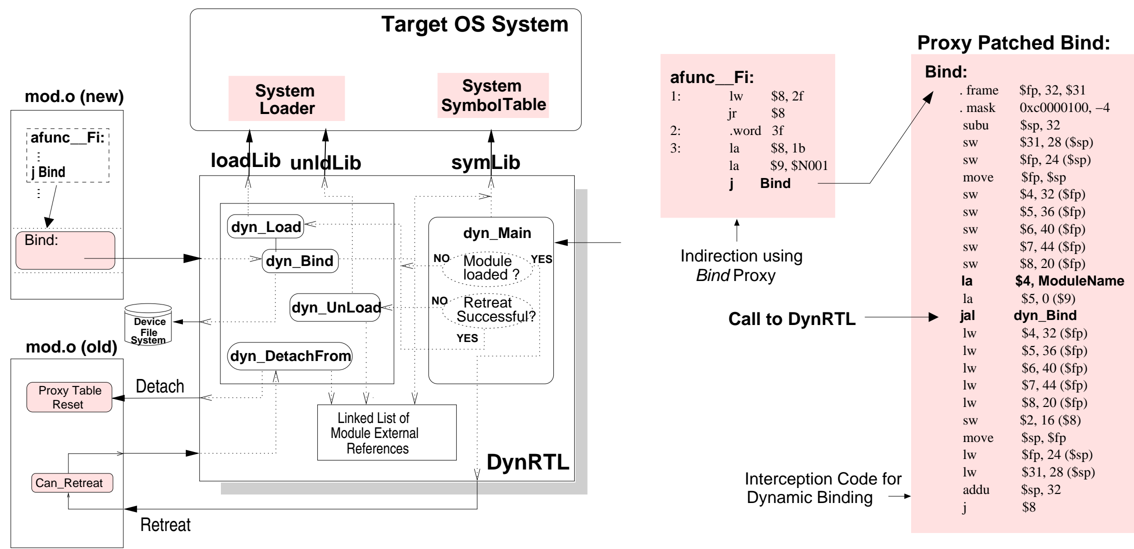

Figure 10: Implementation and operation of the Dynamic Loader (DynRTL) module. Each 'updateable' module in a client device is proxy-patched with a `Bind` function and all external function references are indirected to point (jump) to this function. The function calls DynRTL module (`dyn_bind`) for dynamic binding that resolves the call. (Example code shown for R5 MIPS in VxWorks RTOS platform).

references.

```
DynRTL main entry function:
extern "C" {
        bool                DnLd_Main( unsigned char *dBytes);
};
DynRTL external functions:
extern "C" {
        struct MODULE * dyl_Load( char * filename);
        bool                dyl_Unload( const char * moduleName);
        unsigned int        dyl_Bind( const char  * client_name, char * name);
        bool                dyl_DetachFrom( const char * module);
        void                dyl_DownloadInProgress( bool b);
};
```

Figure 11: Dynamic Loader (DynRTL) functions

Figure 11 shows the functions exported by the DynRTL module. The main entry function for DynRTL is the `DnLd_Main`. Whenever there is a new software update available from the provider server, the download module in the Mobicare client (shown in figure 7) transfers the downloaded data to the DynRTL using `DnLd_Main`. The `DnLd_Main` function performs administrative tasks for operations that are specified in the software update package itself. Some of these operations include:

- Loading, unloading and replacement of existing client modules.
- Installing, deleting and replacing files in the local device file system (data files as well as application software).
- Creating or deleting directories within the local file system.
- Identifying modules that must be loaded and started when the system is booted.

The rest of the functions exported by the dynRTL API are used by the other client modules, as well as the application software in the client device.

Figure 10 shows dynamic update and the binding process of an example module `mod.o`. When there is a dynamic (new) update available for the module `mod.o`, the download module in the Mobicare client first instructs the DynRTL (using `DnLd_Main`) to replace this existing module `mod.o` with a new one. In this case the `DnLd_Main` function checks if the module is loaded in memory and invokes a `retreat` call for `mod.o` by invoking its wrapper `Mod_Retreat` function. The *Retreat* function checks, and if necessary, waits for the module to get into a state where it can be retreated (using `can_retreat` function). It then calls `dyn_DetachFrom` in DynRTL that detaches (deletes) the references of all other client modules to this retreating module (old `mod.o`). Once this module retreats, `DnLd_Main` unloads it from the memory using `dyn_UnLoad` and then loads the new module (using `dyn_Load`) into the memory. In this way any Mobicare client module can be added, or existing module removed or replaced. Replacing a module involves first making in inaccessible to other clients modules by removing its entry point from the symbol symbol table.

5.3 Structure of a Download Package

The structure of a download package gives the layout of the data intended for download by Mobicare client devices. The general structure is a two-level hierarchy, allowing sequential as well as simultaneous replacement of software modules. At the highest level, **groups** are specified and these are processed sequentially. Each group consists of one or more download **entries** that are processed simultaneously.

Data intended for download by the clients is packaged using the **PackBuild tool**. As shown in figure 12 using the tool we can specify operations to be performed with the data package. Operations that may be performed over existing client modules include `remove`, `force retreat`, `replace` as well as options of how and when to effectuate the new data module when downloaded by the client device. For exam-

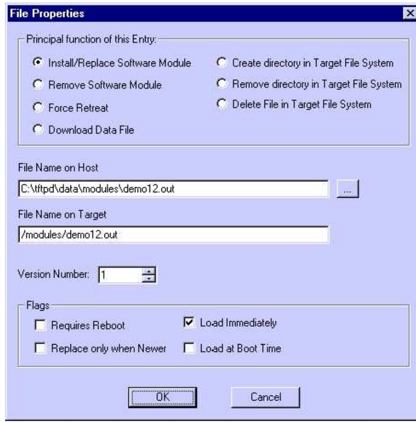

Figure 12: Operation Selection in PackBuild Tool.

ple, the tool specifies several options of how downloaded modules may take effect in the device e.g. `restart`, `load immediately`, `load-when-start` etc. Compression and error checking for the download package is currently not implemented.

6. EVALUATION

We have implemented the Mobicare client functionality in C++. The target operating system used is an embedded real-time OS VxWorks and the Tornado II development tools from WindRiver [27]. Our client hardware consists of an Algorithmics P4032 board with a R5 MIPS, the RM5231 from QED, running at 133MHz. The web server is Apache 1.3.22 running Linux 2.4.21 (single CPU 500MHz, 1GB Memory). Servlet support at the server-end is enabled using JServ.

6.1 Client Measurements

One of the main goals of our work is to provide *hot* software updates applied to the native code of the client device, and do so efficiently. Hence to measure efficiency of our implementation, we adhere to an approach similar to one used in [21]. We examine use of the time and space costs imposed by the design and implementation of our DynRTL. We then compare these overheads with those of the standard Linux DLopen/ELF implementation. (Note that Linux DLOpen/ELF provides dynamic binding but no 'hot' modular replacements.) Our measurements show that the time overheads using DynRTL are competitive. Next, we analyze the space overhead incurred in enabling the dynamic software update functionality using DynRTL and measure other metrics (e.g. code size, size in the memory, etc.) for our client implementation.

6.1.1 Time Overheads

The execution time overhead imposed by dynamic binding using DynRTL for a Mobicare client occurs in three different time scales:

1. **Start-time Overhead (t_s)** – The is the time required for a new module to register its statically-linked code (symbols) into the system symbol table. Note that this overhead is incurred during system initialization (e.g. boot-up) or during 'hot' modular updates.
2. **Run-time Overhead (t_r)** – This is time taken for each external reference of a given module to be indirected from the module's proxy `Bind` function to the `dyn_bind` of DynRTL before being resolved.
3. **Load-time Overhead (t_l)** – This is the time needed by a running program to load a module and link it by executing its initialization `init` function.

Time Overhead	DynRTL (VxWorks)	DLOpen (Linux)
Start-time (t_s)	1.23 ms	–
Run-time (t_r)	0.45 ms	0.21 ms
Load-time (t_l)	0.69 ms	0.38 ms

Table 1: Time overheads for a software update using DynRTL/VxWorks and DLOpen/Linux. (averaged from over 10 runs)

In Table 1 we present the different components that contribute to the time overheads for a sample dynamic software update. We compare these components for our DynRTL implementation in VxWorks RTOS with that of the standard DLOpen implementation in Linux. We find that the start-time (t_s) overhead for DynRTL for the sample update is 1.23 ms and it remains negligible. At the run-time (t_r) the additional overhead comes from using the proxy *Bind* function that provides dynamic binding in DynRTL. Here we find that the run time overhead is more than twice of 0.28 ms offered by DLOpen/Linux implementation. The load-time overhead is also better for the DLOpen/Linux when compared to our DynRTL/VxWorks implementation. The main here difference comes from using the ELF loader in Linux, which is highly optimized and that makes use of the string hash-table in modules for faster look-ups. In contrast our current implementation of DynRTL uses simple linked lists. Note that these differences in the time overheads using DynRTL have **no** noticeable impact on the performance of the different applications.

6.1.2 Space Overheads

All 'programmable' modules meant for dynamic software updates make use of the wrapper layer. The wrapper code enables Mobicare clients to dynamically bind to other client modules as well as *detach* and *retreat* when asked. However, the use of a wrapper layer increases the size of these modules (binary file size) relative to their size after compilation.

The use of a wrapper layer in a module imposes three additional space-related costs: (i) a global proxy address table and indirection for all exported symbols of that module, (ii) 'Bind' as a common proxy to all (exported) functions, and, (iii) use of *Detach* and *Retreat* functions that are patched to that module for enabling 'hot' updates. Notice that 'Bind' and use of *Retreat/Detach* function incurs only a *fixed* space overhead, whereas the size of the global proxy table increases with the number of export symbols (external references) of a given module. Thus the total space requirements using the wrapper code for a given module is the sum of all the above three costs and the corresponding split for our platform is shown in table 2.

When the number of export symbols in a module are reasonable, the increase in the object file size due to the wrapper layer is not much. The total increase in the number

Wrapper Source	Space Cost
Bind Indirection (fixed)	44 bytes (per proxy)
Bind Proxy Code (fixed) + `ModuleName_Detach` `ModuleName_Retreat` (fixed)	496 bytes
Length of Names (variable) for all Proxy Functions	100-200 bytes (per proxy)

Table 2: Space overhead from Wrapper Code.

of bytes for a module file size varies between few hundred bytes to at the most few kilobytes and this is typically insignificant for most embedded applications. Note, however, that the length of mangled names resulting from the use of C++ can become considerable. For example, the use of the standard string class (implemented in C++) and used in our implementation for parsing application protocol messages results in names that are often between 100 and 200 bytes long.

Modules / Metrics	DynRTL (w/Dbg.)	DynRTL (w/o Dbg.)	Mobicare Client. (w/Dbg.)	Mobicare Client. (w/o Dbg.)
Lines of Code	538	415	6122	5358
Bin. File Size	95 KB	44 KB	522 KB	219 KB
Size in Memory	19 KB	5 KB	124 KB	69 KB

Table 3: Measured metrics for the DynRTL and full client-side implementation. (with and without system debug log messages)

Table 3 provides other measured metrics for DynRTL and for the client implementation in Mobicare. It is easy to see from this table that implementation of DynRTL and the Mobicare client incurs minimal overhead in terms of size in the memory and overall binary file size. Figure 13 shows the breakdown of the binary code-size for the different modules used in the client.

6.2 Cellular Link Performance

We conducted tests to provide a realistic picture of cellular link performance and to analyze what impact it may have on the transport of periodic (or on-demand real-time) mobile health data. Our link performance study had two goals. First, we wanted to perform tests to see if cellular links offer data-rates that are feasible for mobile health data transfers (and especially for the uplinks). Second, we wished to evaluate the reliability of cellular links for such health data uploads. Based on the outcome of these tests, we review few techniques that can improve performance.

Our test bed setup consists of a commercial cellular GPRS and UMTS 3G network testbed as shown in figure 14. The client connects to the Vodafone UK's GPRS and UMTS 3G network using a PPP (point-to-point) link. In these tests we use an Ericsson T39m '4+1' handset for GPRS (PPP connection using Bluetooth) with maximum data-rate of 53.2 Kbps and 13.3 Kbps for the downlink and uplink, respectively. For 3G we use a *dual-mode* 3G-GPRS PCMCIA card (Qualcomm chipset) that provides a maximum downlink data-rate of 384 Kbps and an uplink data-rate of 64 Kbps. Link-layer (ARQ) retransmissions for both GPRS and 3G network is kept enabled (network default).

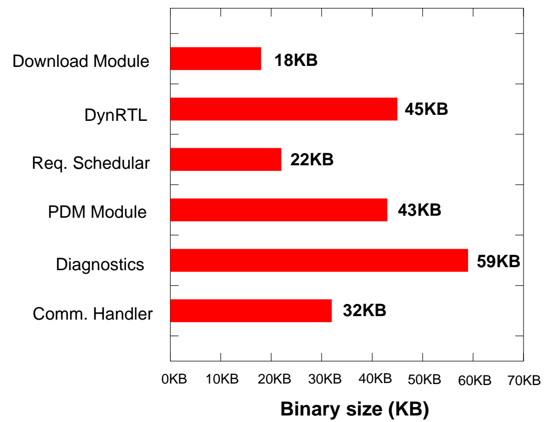

Figure 13: Breakdown of Client Code-size.

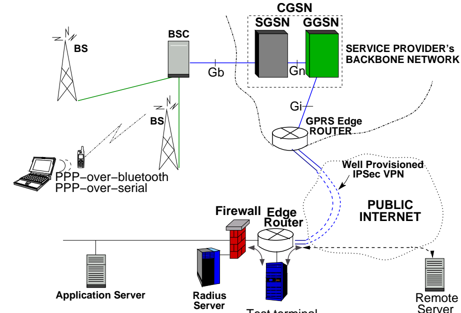

Figure 14: Test Bed Set-up.

6.2.1 *Link Performance Tests*

We performed tests to measure packet latencies as well as up/down link bandwidth using TCP. In these tests we measured the aggregate up and downlink throughputs available using GPRS and 3G networks simultaneously. During these tests any incidence of packet loss and re-ordering was noted. To understand the steady-state network behavior we selected a reasonably large file size (> 500KB) for these transfers. The traces collected were then analyzed using `tcptrace` [8]. It is to be noted that research has already investigated performance of cellular (e.g. GPRS) downlinks thoroughly. However, in this study, we also evaluate uplink performance in UMTS 3G that is crucial for health data uploads in Mobicare.

Figure 15 (a) and (b) shows the measured uplink and downlink throughputs for UMTS 3G and GPRS networks, respectively. As can be seen, TCP over 3G achieves good data-rates for up/down link and close to the maximum offered data-rates. We find from figure 15 that UMTS 3G offers up/down link data-rates that are an order magnitude higher than that of GPRS. This can have interesting implications for Mobicare. For example, an uplink transfer of 50 KB file in 3G may take around 8 secs while the same file takes more than 80 secs to upload in GPRS! We can also see that both GPRS and 3G show significant, often sudden, throughput fluctuations as can be observed from figure 15 (b). The measured round trip times (RTT) observed for the UMTS 3G links was between 250-350 ms, which is lower

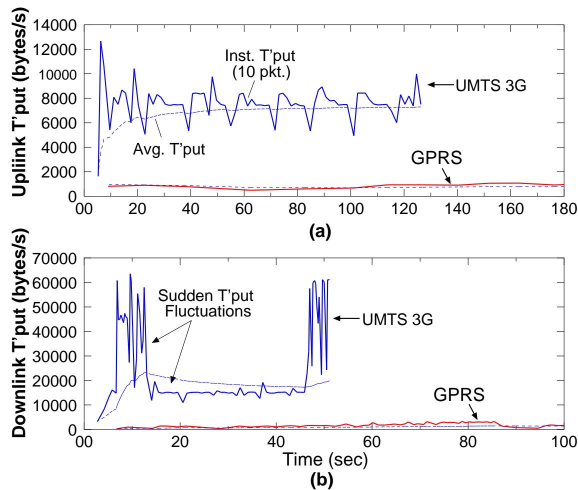

Figure 15: Plots shows (a) uplink and (b) downlink throughputs measured over GPRS and UMTS 3G networks. (Cellular links are prone to sudden throughput fluctuations.)

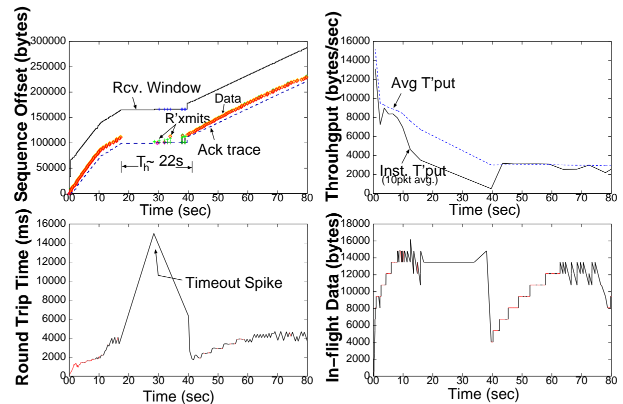

Figure 16: Plots capture impact of the link-layer 3G→GPRS handover during data (file) upload with (top-left and clockwise) (a) time-sequence plot, (b) throughput plot, (c) outstanding-data plot, and, (d) round-trip time plot. (UMTS 3G links are currently prone to such frequent handovers to GPRS.)

than the average RTT (around 800 ms) offered by GPRS.

During file upload tests, we observed that UMTS 3G links are also prone to frequent handovers to GPRS. Using a dual-mode 3G/GPRS card, we found that 3G link would typically handover to GPRS and back. Figure 16 shows one such example that captures the impact of the link-layer 3G→GPRS handover (link-layer as there is no change of the network IP address) with a file upload in progress. Here, we can observe that such handovers can potentially impede data transfer (during uploads) for a significant amount of time. The total handover interval in this case comes to around 22 secs, during which time a number of TCP retransmissions occurred before data transfer could resume.

These experiments show that, while although GPRS/UMTS cellular links offer data rates that are sufficient for mobile health data transfers (both up and downloads), they still pose many challenges to overcome. Because of the 'health-critical' nature of certain mobile medical applications, link reliability for efficient and timely data delivery is crucial and therefore comes as an important metric for health applications.

6.2.2 *Need for Data Adaptation and Diversity.*

From the previous section we find that cellular links are bandwidth limited and links exhibit wide asymmetry also evident from figure 15. Therefore compressing mobile health data can be very useful to reduce the amount of data transferred and enable faster and efficient data delivery over cellular links. For example, 2-lead ECG signals captured with 12-bit resolution gives a data-rate requirement of around 60 Kbps. However, by using compression the data-rate requirements can be effectively reduced to around 8 Kbps, an improvement by a factor of *at least* 3. Thus Mobicare can benefit from data adaptation as adaptation is the key to mobility [11].

An important and useful technique that can overcome link fluctuations and outage-like situations is by exploiting the simultaneous coverage access available from multiple wireless cellular providers. The idea here is to exploit *diversity* from multiple such cellular access links, thereby reducing the overall probability such link-outages that currently plague mobile cellular systems. MAR commuter router [24] is one such system that can efficiently aggregate bandwidths by exploiting cellular diversity from multiple wireless links simultaneously, thus eliminating the chance of a link-outage or potential performance degradation due to severe link fluctuations during data transfers. Mobicare clients can also benefit using a highly reliable uplink, by aggregating bandwidths and exploiting link diversity similar to the MAR system.

7. DISCUSSIONS

In this section we discuss some other important issues relevant in the context of Mobicare.

7.1 Device Reconfigurability

Mobicare offers a modular architecture (instead of a monolithic system) that enables inter-module references to be resolved at run-time; modules can be replaced without consequences to other modules. Thus device reconfigurability comes as a natural benefit using a modular system architecture. Mobicare implements dynamic loading functionalities that are partly similar to, and partly to exceed those that are found in dynamic link library (DLL). Traditionally, DLLs are loaded together with the applications that use it, whereas client modules in Mobicare are loaded-when-needed. A *loaded-when-needed* architecture is quite similar to the Microsoft's Component Object Model (COM) [22]. However, in contrast to COM the concept of `QueryInterface` is not used. This makes the system slower (the process of finding a required function reference is more elaborate) but at the same time more flexible (no grouping of functions is enforced).

System modularity in MobiCore introduces no special requirements on the modules that populate the system. Whereas COM modules must be built according to the strict and complex rules of the COM architecture, Mobicare offers a much simpler architecture. The important benefit comes using an

arbitrary third-party module, built with a monolithic OS model in mind, that can now be proxy-patched using the wrapper tool and ready for use in dynamic environments. This feature of dynamic run-time modular replacements is unique to Mobicare.

7.2 Security Issues in Mobicare

On account of limited system resources, current clinical sensors may be rendered ill-suited for a basic implementation of security protocols. Hence a shared-key infrastructure for clinical sensors appears less practical due to the computational overhead involved. Nevertheless, security issues are also generally less restrictive for a patient sensor network. This is because such sensors are expected to fulfill certain clinical (dedicated) task or functionality. Such sensors are typically confined to the body sensor network, and only in limited cases, may involve sharing key with a mobicare client device.

Client-server security can be addressed by implementing session layer security. Protocols such as WAP-based Wireless Transport Layer Security (WTLS) protocol can be used to provide privacy, data integrity, and authentication over the cellular link [9]. WTLS is readily available in most WAP-based mobile phones. Moreover, it closely resembles Secure Socket Layer (SSL)/Transport Layer Security (TLS) protocol, yet is optimized for use over low bandwidth wireless links and suits resource-restricted mobile devices. Furthermore, application servers may also introduce security vulnerability by exposing server's IP address to the clients. This potential danger can be limited to some extent by ensuring that only clients from the wireless network connect to the server, thus, limiting the load an individual client generate in an attempt to starve others.

7.3 Radio Over-exposure

Radio exposure caused by surrounding radio devices may introduce additional health-related risks. However, it is unclear to what extent this exposure may exacerbate the risk towards a patient's health. For example, Ericcson suggests heart pacemaker users to keep a distance of at least 15 cm between a cell-phone and the pacemaker [1]. Wearable wireless sensor devices may also introduce additional risks to a patient's health due to unwanted electro-magnetic radiation. Important factors that may influence exposure coming from radios are frequency, transmitted power level, modulation, and distance. An indepth study is required to critically understand and investigate issues so that such risks could be identified and mitigated. A recent study conducted by NPRB [15] in the UK recommends restricting radio exposure to a level below a certain threshold. This, it says, protects against the potentially adverse effects on patients susceptible to electrical stimulation.

8. RELATED WORK

Researchers all over the world have started new projects to investigate the potential in mobile healthcare [2, 14, 7]. Interestingly, much instigation in this area is stimulated by the rapid advances made in wearable computing, sensor networks and mobile communications. We discuss these issues in the context of Mobicare.

Clinical trials to gain insight into how medical systems may help patients evolve in mobile settings were conducted in the mid-nineties by the National Library in Medicine for Mobile Telemedicine project [16]. Key lessons learned from these tests were that: (i) achieving reliable, high-bandwidth wireless data communications is difficult, (ii) transmission of critical patient data during emergencies can make significant difference in patient outcomes, and, (iii) remote patient diagnosis may be difficult. Note that most limitations as seen in the Mobile Telemedicine Project have been overcome by the advances made in the areas of clinical sensors, wearable computing and mobile communications. Collective progress in these areas make mobile medical care a reality.

The European Mobihealth project [2] is one such example that inspires from the worldwide introduction and the deployment of high-speed cellular technology. It aims at developing and testing new mobile value-added services in the area of healthcare, thus bringing healthcare to the patient. Health care services offered in Mobicare are similar in spirit to that in Mobihealth – both aimed at solving mobile monitoring needs for patients through cellular technology.

Dedicated clinical sensors have long been used (though rather restrictively) in various medical settings. Such sensors are examples of *small-form* wearable devices that accomplish certain dedicated tasks. However, sophisticated wearable device such as the IBM's Linux Wrist Watch [18, 12, 13] go way beyond the potential of the such wearable clinical sensors. The watch packs many useful features: Linux OS, X11, VGA graphics, Bluetooth and IrDA wireless connectivity into a small, wearable, wristwatch device. We believe that such wearable devices combine the necessary functionalities to accomplish the different tasks in medical care settings. Therefore, barring few simple modifications to this wristwatch device, porting Mobicare client functionality should be straightforward. For patients a wearable device like the wrist-watch will not only monitor their 'health-critical' data, but can also collect, store and perform periodic uploads with the health servers. Thus a wearable wristwatch device comes as an integral component of any mobile healthcare systems.

The *Personal Server* from Intel [23] is very useful and innovative concept for mobile healthcare. The Personal Server is a mobile device that can readily store and access data and applications, and wirelessly communicate with the local environments. Such a device has practical use in mobile health care settings for: (i) high-density data storage, (ii) power-efficient computing, and, (iii) short-range wireless (Bluetooth-based) connectivity. Thus the personal server can be made to work much like a patient 'health hub'.

Mobicare benefits from major sensor research projects like the CodeBlue project at Harvard University [25, 14]. The CodeBlue project offers many useful features for patient sensor monitoring: (i) a robust co-ordination and communication substrate across sensor devices, (ii) a publish/subscribe model for data delivery, and, (iii) adhoc networking for "mesh-like" secure data connectivity for sensor nodes. Using the MICA2 mote (originally designed at UCB [19]), CodeBlue has developed a wireless pulse oximetry sensor and electrocardiogram (ECG) sensor that can continuously monitor and record vital sign and cardiac information from large number of patients. CodeBlue offers many useful patient network centric services. However, these services compliment those offered by Mobicare and both can co-exist to provide very effective mobile medical monitoring.

Mobicare shares many of the goals and objectives with the Patient Centric Network (PCN) project started at MIT [7].

The PCN team is currently developing a prototype that addresses various service-specific issues for the patient sensor network. The system will consist of software components running on general purpose computers and networks that will link users with a variety of medical sensors and actuators. The goal in PCN, similar to that in Mobicare, is to accelerate innovation, decrease cost, and improve the clinical quality of medical care. Thus PCN can benefit from Mobicare with a programmable service architecture and mobile medical monitoring.

Related research has investigated programmable architectures in some other contexts. For example, the work in [20, 26] investigates the programmability at the physical layer based on digital signal processing techniques. The programmability at the physical layers allow new functionality to be incorporated such as modulation, equalization, channel coding etc.. Programming quality of service (QoS) in mobile networks has been well-explored in [10, 17]. [10] proposes a programmable MAC framework, while [17] presents Mobiware middleware that exerts QoS control over wireless access networks for adaptive mobile multimedia applications. Mobicare inspires from such programmable architectures to help accelerate deployment of new mobile health services and applications.

9. CONCLUSIONS

The vast opportunity in 'point-of-care' access and the capture and transmission of patient information will continue to drive the healthcare industry towards increased mobility. The importance is in the shifting awareness that mobility increasingly refers to – the mobility of devices, the healthcare providers (health 'outsourcing') and of the patient (users) themselves. Mobicare is the first architecture that leverages 'point-of-care' medical access to provide important benefits:

- **Quality Health Care:** Mobicare enables continuous, round-the-clock monitoring for chronically-ill patients. This not only improves the quality of patient care, but also reduces relapse rates, overall hospitalization period and costs.
- **Programmable Architecture:** A programmable architecture allows for easy introduction, configuration and customization of diverse medical sensors to a patient sensor network. Client devices can update with new medical features, applications and services that are tailored to meet the requirements of patients and the health providers.
- **Flexible Service Components:** Services in mobicare can dynamically self-activate, (re)configure, update and can be customized to suit (medical) monitoring needs of the patient and the health providers. These services effectively address the requirements of the patient's medical monitoring needs – the most significant challenge in mobile healthcare.
- **Medical Systems Integration:** Mobicare enables medical systems integration with full control for a body sensor network. Health providers can have continuous access, control, and configuration of body sensors using 'always-on' cellular connectivity.

Mobicare enables heathcare personnels to be able to timely access, review, update and send patient information from wherever they are, whenever they want. These factors, coupled with current and expanding healthcare applications, will further enhance personal health and ultimately population health thereby increasing productivity.

Other than the population well-being, a sound health care infrastructure can commensurately impact the economic health of a nation. In terms of overall economics healthcare constitutes a significant fraction of a nation's overall Gross Domestic Product (GDP). Therefore, large-scale deployment and use of such mobile healthcare infrastructure may lead to significant economic benefits and cost savings for a nation. In this way mobile healthcare can have both long-term social as well as economic implications for a nation.

10. REFERENCES

[1] Ericsson Health. http://www.ericsson.com/about/health/.
[2] EU MobiHealth Project. http://www.mobihealth.org.
[3] Health Frontier Inc. http://www.healthfrontier.com.
[4] Linde Medical Sensors AG. http://www.linde−ms.ch.
[5] Nonin Medical Inc.. http://www.nonin.org.
[6] Numed Holdings Ltd.. http://www.numed.co.uk.
[7] Patient-Centric Network at MIT/LCS. http://nms.lcs.mit.edu/projects/pcn/.
[8] tcptrace. http://www.tcptrace.org.
[9] WAP Forum. http://www.wapforum.org.
[10] A.T. Campbell, M. E. Kounavis and R.R-F. Liao. Programmable Mobile Networks. *Computer Networks and ISDN Systems* (1999).
[11] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn,K.R. Walker. Agile application-aware adaptation for mobility. In *Proc. of SOSP* (1997).
[12] C. Narayanaswami, et al. IBM's Linux Watch: The Challenge of Miniaturization. *IEEE Computer* (2002).
[13] C. Narayanaswami, M. Raghunath, N. Kamijoh, T. Inoue. What would you do with a Hundred MIPS on Your Wrist? In *IBM Research Report (RC 22057)* (2001).
[14] D. Malan, T. Fulford-Jones, M. Welsh, S. Moulton. CodeBlue: An Ad hoc Sensor Network Infrastructure for Emergency Medical Care. In *Proc. of International Workshop on Wearable and Implantable Body Sensor Networks* (2004).
[15] Documents of the NRPB. Review of the Scientific Evidence for Limiting Exposure to Electromagnetic Fields (0-300Ghz). *National Radiological Protection Board 15*, 3 (2004).
[16] Final Report – Mobile Telemedicine Testbed For National Information Infrastructure. National Library of Medicine, Aug 1998. Project N0-1-LM-6-3541. http://collab.nlm.nih.gov/tutorialspublicationsand materials/telesymposiumcd/bdmfinal.pdf.
[17] G. Bianchi and A. T. Campbell. A Programmable medium access controller for adaptive Quality of Service Control. *IEEE JSAC, Spl. Issue on Intelligent Techniques in High Speed Networks* (2000).
[18] IBM Linux Wrist Watch. http://www.research.ibm.com/WearableComputing/.
[19] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister. System Architecture Directions for Networked Sensors. In *Proc. of ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2002).
[20] J. Mitola. Technical Challanges in the globalization of Software Radio. *IEEE Communications Magazine* (1999).
[21] M. Hicks. Dynamic Software Updating. *PhD Thesis, Department of Computer Science, University of Pennsylvania* (Aug. 2001).
[22] Microsoft COM. http://www.microsoft.com/com/.
[23] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar and J. Light. The personal server: Changing the way we think about ubiquitous computing. In *Proc. of the 4th international conference on Ubiquitous Computing (UBICOMP)* (2002).
[24] Reference deleted for double blind review.
[25] T. Fulford-Jones, G-Y. Wei, M. Welsh. A Portable, Low-Power, Wireless Two-Lead EKG System. In *Proc. of 26th Annual International Conference of the IEEE EMBS* (2004).
[26] V. Bose, M. Ismert, W. Wellborn and J. Guttag. Virtual Radios. *IEEE JSAC. Spl. Issue on Software Radios* (1998).
[27] Wind River Inc. http://www.windriver.com.