Chapter 8
Mobile OS and Application Protocols

8.1 Introduction

Protocol level details on how a mobile network such GSM/GPRS, WLAN or WPAN functions is of no concern to most of the users of mobile technology. An ordinary user is interested for ubiquitous access of information over mobile devices. In other words, the users would expect these mobile access protocols to work seamlessly giving a feeling that the access is as easy as it is on wired networks. For instance, a mobile user will be interested to access information and interact with services provided over the Internet. Some of the popular form of interactions include Web browsing, messaging service, SSH, etc.

Mobile devices maintain connectivity to the rest of the world through various cellular based wireless networks such as GSM, GPRS, 3G, LTE, Bluetooth and IEEE 802.11x. The network connectivity is intermittent due to inadequate coverage, signal interferences, and low bandwidth. So mobile OS are designed to manage service interruption and ad hoc reconnection attempts. When reconnecting back to network, a device may use a new wireless channel. From the prospective of use case scenarios, many of the utility services accessed by the mobile users are location dependent. Therefore, Mobile OS should be able to opportunistically combine location service from different networks, GPS and compass for the benefit of applications. Normally, a user is careful about losing mobile devices, yet small devices are prone to accidental mishandling as well as misplacement. Though a mobile OS cannot provide a complete security solution, it should at least incorporate provisions by which services can be stopped from unauthorized accesses. Increasingly, mobile devices are getting more powerful and resource rich. Research on mobile grids have explored way to harness aggregated computational capabilities of mobile devices. However, this research still in infancy and remains within confines of academic and research labs. Considering OS issues are just too many, the discussion on mobile OS in this chapter is organized around the basic features, focusing on the minimality and the essential components.

In addition to mobile OS, our aim is to cover three protocols, namely, Mobile IP, Mosh and WAP. Mobile IP [1] is a modification of wire line IP at the Internet level

© Springer Nature Singapore Pte Ltd. 2017
R.K. Ghosh, *Wireless Networking and Mobile Data Management*,
DOI 10.1007/978-981-10-3941-6_8

which allows a user to receive messages irrespective of his/her home registration area. A wire line IP assumes a fixed unique attachment point for a node. Messages can be delivered over the Internet by tagging them with corresponding IP addresses of the nodes. An IP address with a port number identifies a unique connection from among several concurrent services that may be running on a node. A route from node S to a destination node D may not be the same as the route in the reverse direction. Between two communicating end points, TCP [2] distinguishes one internal session from another by using IP addresses of the respective endpoints along with the demultiplexing selector for each session. IP address allows to identify a route to a destination. The only other thing that may influence routing of packets is network congestion due to excessive traffic.

SSH [3] command is quite familiar in the context of accessing remote computers from any where. SSH works beautifully when network delays, and packet losses are within tolerable limits. However, SSH does not support IP roaming. It works on a terminal emulator which processes one character at a time. This implies SSH traffic from a server consists of a lot of packets having very small payloads. All the echos or in-line editing are performed by a remote computer known as the server. Therefore, it becomes completely useless when packet losses occur. In other words, SSH cannot handle intermittent network connections or networks experiencing long delays. Wireless networks, on which mobile services are based, often experience long delays and intermittent connectivity. On such an environment SSH becomes ineffective and useless. Mosh [4] or mobile shell is a remote terminal application which not only can handle IP roaming, but also supports client side predictive echoing and line editing. It does not require any change in the server side software for running applications. Mosh is essentially a shell application that can work without requiring a connection to be maintained over flaky links to support a user's movements. Since SSH uses TCP, preserving terminal session becomes an issue for supporting IP address roaming as the user moves about. Mosh on the other hand uses UDP.

Wireless Application Protocol (WAP) [5] is a framework for accessing many value added services over Internet and enterprise intranets using mobile devices. Apart from known hurdles in information access over wireless interfaces, small form factor of mobile devices calls for a whole new approach to the presentation of information. WAP framework was formulated as an industry wide global standard to bring development of all applications over mobile wireless networks. The targeted applications include microbrowsers, scripting, email, chat, instant messaging service among others. So, WAP is not just a single protocol but a protocol suite. Accessing Internet resources on hand held wireless devices becomes efficient through WAP.

It is expected that the discussion in the chapter will provide enough pointers to create applications which can interoperate irrespective of communication technologies and a whole range of hand held devices with different capabilities as well as form factors.

8.2 Mobile OS

A Mobile Operating System (MOS) is designed for mobile devices such as PDAs, tablets, and smartphones. Although laptop computers are also portable mobile devices, they do not belong to the category of devices which can be classified as mobile phones. Usually, laptops run on desktop OSes. Unlike smartphone, laptops are neither equipped with modems nor possess a variety of sensors. However, some variety of ultra thin hybrid laptops have touch screen support, foldable display, front/back cameras, detachable or sliding keyboards, pen support, and a few sensors such as accelerometer, ambient light sensor, gyroscope, GPS and compass. When equipped with modems, these devices can also function as tablets.

Mobile OSes have been implemented taking various resource constraints of mobile devices into account. These include limited energy and limited computing capabilities, small form factors and non conventional input/devices. It forces implementation to be oriented more towards gestures, touch and voice based interactions. Mobile devices maintain connectivity to the rest of the world through various cellular based wireless networks such as GSM, GPRS, 3G, LTE, Bluetooth and IEEE 802.11x. The network connectivity is intermittent due to inadequate coverage, signal interferences, and low bandwidth. So mobile OSes are designed to manage service interruption and ad hoc reconnection attempts. When reconnecting back to network, a device may use a new wireless channel. From the prospective of use case scenarios, many of the utility services accessed by the mobile users are location dependent. Therefore, Mobile OS should be able to opportunistically combine location service from different networks, GPS and compass for the benefit of applications.

8.2.1 Smartphones

Among various mobile devices, smartphones dominate mobile device market. Most people, specially those in age group of 18–35 years, tend to access mobile services through smartphones. So, the discussion on mobile OS remains incomplete without a reference to smartphones.

Smartphones have brought in a sea of changes and convenience in daily lives world over. An estimate [6] projects that the smartphone users base will reach 2.87 billion by 2020. As of 2016, there are about 2.1 billion users of smartphones. The rate of growth is more in emerging economy than in rich economy. The reason is attributable to the fact that smartphones serve endpoints for maintaining both connectivity and control over contents and services even if the conventional communication infrastructure is not very good. Therefore, people belonging to low and middle income group depend heavily on smartphones for their livelihood.

8.2.1.1 OS as an Abstraction Layer for SoC

Battery life is a critical aspect in determining the user's experience on a smartphone. Due to portability reasons, the phone battery should be slim, light weight and a long lasting power pack. As opposed to an ordinary cell phone, the design goals of a smartphone are: thinner device, slimmer battery, bigger screen, faster CPU, and larger memory size. Some of the design goals are mutually conflicting. To achieve these goals, smartphones use SoC (System on Chip) [7] processors. SoC defines an integrated design that includes CPU, GPUs, a number of co-processors, and memory. Roughly speaking, SoC is equivalent to the mother board of a desktop computer.

Our focus in this chapter is not smartphone architecture. However, the performance and the speed of execution of applications depend on the architectural parameters of a computer. A user's experience is not only determined by the applications that run on a smartphone but also how fast the applications can run.

An operating system leverages hardware and provides software abstractions that can be exploited by the developers to provide rich user's experience. In abstraction, a SoC design can be viewed as a collection of network of modules. Each module provides a specific service. The connection between services is supported by bus or network. The OS requirements as follows [8].

- In order to take take full advantages of modularity of SoC architecture, OS design must be modular. To provide the best performance, it should be possible for a developer to abstract out the application design according to the knowledge whether a particular OS functionality is implemented in hardware or software.
- Modular approach in the SoC design means, the hardware designers may also regroup SoC components even during the development process. Smartphone OS design should be able to adapt to these changes without requiring a redefinition of hardware abstractions presented to the application layer.
- If a specific hardware functionality is defined by overly generic abstractions, then it might prevent an application from exploiting the advantages of that the hardware component through the specific OS abstraction.
- OS should also provide for the protection and the concurrency control. It implies different parts of the OS and the application (e.g., interrupt handlers, drivers, application code, etc.) cannot (expect to) interfere with each other.
- Due to the application specific nature of SoC, it will be necessary to port the OS to many different platforms. The OS and application code should, therefore, be highly portable. There should not be any cross dependency involving different functions of the OS. In other words, each OS function must be well encapsulated.
- Since, most SoC systems have real-time and dependability requirements, it is desirable that OS abstractions are amenable to verification of correctness and temporal analysis.

In view of the above, a micro kernel based approach to the design of OS appear to be ideal for (reconfigurable) SoC based devices.

8.2.1.2 OS as a Interface Layer for Applications

At the highest level, the design and the implementation of an OS is dependent on type of the system and the hardware. At a level lower than hardware abstraction layer, OS must be designed meet the requirements of both the users and the system. The user requirements stem from the convenience in use, reliability, security and speed. In contrast, the system goals are guided by ease of implementation, maintenance, flexibility, performance and efficiency as explained in Sect. 8.2.1.1.

A number of studies have made been made to explore of the diversity in the use of smartphones [9]. These studies primarily indicate two different aspects in the use of a smartphone, viz., (i) convenience, and (ii) innovation. In an average, about 22 Apps are installed by an Android user while about 37 Apps are installed by Iphone user [10]. The voice based communication over a smartphone is not substantially different from the level it is found with a ordinary cell phone. In an average, the number of voice calls per phone is around 5.7 calls per day [11]. The use of a smartphone is found to be oriented predominantly for non-voice based applications [12]. All the user sessions concerning applications or computation except for voice calls, are referred to as non-voiced. Texting forms a major part of the non-voice usages. According to one study, each user in an average access SMS or instant messaging app 11.2 times per day [11]. Still there is substantial diversity in the use of non-voice Apps. The types of usages can be classified as follows [11]:

- Recreational
- Instant messaging
- Communication and Internet
- Transport and travel
- Banking and m-commerce
- Knowledge and work

A few knowledge user (those having background in CS) may use a smartphone for App development, but this type of users are only a few. Considering the usage patterns and the diversity of use, two the major concerns in wide spread use smartphones are: (i) energy drainage, (ii) resource scarcity. Some of the resource limitations include low CPU capabilities, display, memory, user interfaces, etc.

8.2.2 Difficulties in Adopting Desktop OS

From the point of view of OS kernel, the difference between desktop OS and Mobile OS is little. About 250 patches (amounting to 3MB of code) were defined for the Linux kernel to tailor it for Android [13]. These patches are primarily aimed at fulfilling the requirements of certain basic functionalities, and overall conservation of resources. Mobile devices should enforce tight power management features, support non-conventional I/O devices. For example, the inputs may be gesture based or through keypad, touch screen, camera, and audio. For portability reasons, a mobile device has a small form factor, and depends on the battery for its operation. So, the conservation of resources is extremely important. Some of the enhancements includes incorporating of new drivers [14]. For example, in Android, a Alarm driver manages timer, which wakes up the device from sleep. Ashmem drivers manages sharing of memory at kernel level. Mobile devices normally support only a small flash memory. So, sharing data and services is important when a device runs multiple applications. Binder driver is needed for interprocess communication. Interprocess communication is supported through the use of shared memory. A service which registers as a IPC service does not have to keep track of different threads in execution. The binder monitors, handles and manages all the threads, and facilitates communication between the processes through shared memory. The binder also takes care of synchronization between the processes.

Mobile OS generally does not kill an application. All other applications opened by a user continue to run in the device even after the user switches to a new application. A running application is killed only when the device runs out of memory. Even then, the state of the application is saved to enable quick launch. The desktop version of Linux distribution comes with a huge GNU C library. It provides library routines as per ISO standard C specification for programming. This library is far too large for mobile phones. For example, glibc also provides Native POSIX Thread Library (NPTL). NPTL offers high performance, but only suitable for the computers having adequate resources as they require large disk foot prints. For a device with limited resources, NPTL is not suitable. Only a few threads may run concurrently in a mobile device. So, many of the features like mutex conditional variables are considered as unnecessary [14].

8.2.3 Mobile OS Features

Three major responsibilities of an OS are memory management, process management and inter process communication. The implementation of virtual memory is a major requirement of memory management for a desktop computer. In a mobile device, implementation of virtual memory is not needed. In the event of a device hitting a low memory level, the running applications are simply killed in increasing order of their priorities. Therefore, many complications including swapping which are necessary for the implementation virtual memory could be eliminated. However, a mobile OS should implement highly secure memory protection schemes to ensure applications do not access each others data or address space.

Normally, for memory protection, desktop computer employ Address Space Layout Randomization (ASLR) technique [15]. It randomizes base point of the stack, the heap, the shared libraries and the base executables. So, it becomes difficult to guess the foot prints of a code residing in the memory. As a result, network based control hijacking attack become difficult. Mobile OS spends time to minimize the boot time, the power consumption, the memory foot prints and the application launch time. Typically, the libraries are pre-linked for this purpose. Sometimes even library addresses are hard-coded. Furthermore, due to security reasons the file system on device is mounted in the read only mode. So, it is not possible for a binary editor to modify the image of a device memory. In [16] a modified randomization scheme is proposed for the pre-linked code in a mobile devices.

Generally, mobile OS employs user initiated event driven interactions. Therefore, the client-server session based IPC is an ideal choice for the communication mechanism among user space components.

8.2.3.1 Kernel

Mobile OS typically employ micro kernel approach for provisioning OS services. The approach does not just mean that the kernel is of small size, rather it provides a set of essential OS services which are used by a different set of other processes that team up together to provide high level OS services to the applications. The system level services such as file system, network interface and TCP/IP are implemented as user libraries, and do not need privileged executions. Access to these services are coordinated by the kernel through a client-server mechanism. The kernel is directly responsible for the management and the protection of memory for both the user applications and the applications providing the OS services. The kernel also provides few other core system services such as process management, scheduler service, and driver modules. Since, high level OS services are implemented as non-privileged processes, it is possible to extend the OS services using APIs in the user written applications.

8.2.3.2 Execute in Place (XIP)

Diskless devices support execute In Place (XIP). XIP is used in smartphones in order to execute programs directly from storage without copying them into the RAM. Mobile OS leverage XIP to optimize the use of memory during the execution of an application. XIP not only speeds up execution, but also indirectly extends the amount of shared memory. Therefore, the total memory requirement is also reduced.

8.2.3.3 Application Framework

Mobile users have options to avail the services offered via a plethora of mobile Apps. Many of the Apps are designed for the convenience in transactions related to banking, travel and electronic commerce. A rich set of Apps also exists for social networking, gaming, fitness, healthcare, and multi-media entertainment. There is a huge scope for abusing the Apps route with malicious intent to harm the users of the mobile devices. Therefore, mobile security poses big challenge. Sandboxing is a simple idea to isolate the Apps and prevent any malicious app from monitoring the activities or the execution of another App. Sandboxing works as follows. A user ID

(UID) different from the invoking user is assigned to an application at the time of installation. The access control for each application's UID are statically assigned at the time of installation. Mobile OS enforces the discretionary access control regime according to the UID of each application when it executes.

Although Apps run in a sandbox environment, the application specific data are often stored in shared resources. Therefore, it may be possible for one App to access the data of another App. By collecting footprints of various data items, it may also be possible to make intelligent guesses about the transactions through data mining and other learning techniques. Therefore, data caging is used in conjunction with sandboxing to isolated application data.

8.2.3.4 Security Issues

One of the simple techniques is filtering out unverified Apps and controlling the release of third party Apps. Employing the process of App vetting before the release of an App could be a practical solution in this respect. For Android system, Google developed Bouncer testing mechanism [17]. It is a well known way of app vetting. However, malicious App developer may use evasive techniques to dodge the App vetting process. One simple way is to employ obfuscation techniques whereby, a malicious App uses the finger prints of dynamic analysis of the vetting system and avoids the detection of malicious code. So, when the same App subsequently runs in a actual device the malicious code get triggered.

Antivirus is another complementary method of used to ensure security. However, antivirus softwares have limitations [18]. An antivirus program typically scans the code and package information in an App, but cannot understand the runtime behaviors. Monitoring of runtime behavior is necessary for better detection method. One possible idea could be to instrument Apps for monitoring their runtime behaviors. So, Apps need to be repackaged. However, application repackaging is not supported by most systems. Another approach could be to identify all sensitive calls that an application makes to Java API, and replace them by special functions that monitors the behaviors. However, this approach is possible only for the Apps written in Java. Any other approach will require changes in different layers of OS stack.

8.2.4 Mobile OS Platforms

A variety of Mobile OS platforms exists and used. Notable among these are:

- J2ME
- Palm OS
- Symbian
- BlackBerry OS
- Android

- Windows Mobile OS
- iOS

Any attempt to provide a comprehensive review of the internals of mobile OSes is beyond the scope of this book. Therefore, our focus in this chapter is restricted to review of four widely used mobile OSes, namely, J2ME, Symbian, Android and iOS. The choice has been dictated by the historical aspects of the development of mobile OSes as well their popularities among the users.

8.2.5 J2ME

Sun Microsystems released three different editions of Java 2 platform to meet the entire range of computing requirements from enterprise servers to small personal devices. Java 2 Enterprise Edition (J2EE) provides the server based solutions for the computing needs of enterprises. Java 2 Standard Edition (J2SE) is cut out for the software solutions on desktops and workstations. Finally, Java 2 Micro Edition (J2ME) addresses the computing requirements of Java enabled hand held portable devices such pagers, personal organizers, PDAs and phones. The Java big picture is illustrated by Fig. 8.1.

A set of specifications defines a J2ME platform for programming of small personal devices. It consists of three building blocks, namely, Configuration, Profiles, and Optional packages. Optional packages define a set of APIs for supporting some additional behaviors and common features. Bluetooth and JDBC may be present as a part of optional packages. Configuration and Profile are two important building blocks of the J2ME architecture. Each specification set is referred to as a profile.

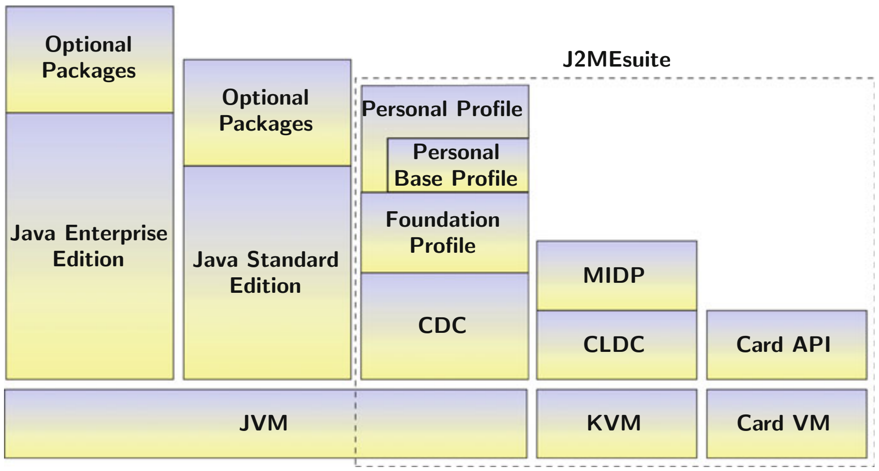


Fig. 8.1 Java 2 platforms—the big picture

Each J2ME profile is built by extensions of the basic capabilities of the configuration layer in J2ME stack. The configuration and the profiles for a device are developed on the basis of the hardware features of the device and the targeted for the use of the device.

8.2.5.1 Configuration

Configuration layer defines the abstractions for generic mobile devices having similar capabilities. Configuration layer is implemented over the host JVM that relies on the host OS as shown in Fig. 8.2. Configuration layer is closely inter-woven with JVM in order to capture the essential capabilities of each family of devices. The members of a family of devices are distinguished by the specifications defined by the profile layer. A profile specification is implemented by a set Java class libraries. To meet the requirements of a new device type, either a set of new class libraries should be added or the existing libraries should be modified and extended suitably.

There are two different types of configurations, CDC (Connected Device Configuration) and CLDC (Connected, Limited Device Configuration). CDC configuration is meant for virtual machines hosted on a desktop or on a laptop. CDC consumes higher memory resources compared to CLDC. CLDC requirements are supported through a small virtual machine called KVM defined on the top of JVM. KVM is designed for Java programs that require a total memory of only few kBs. KVM can be viewed as a restriction or contraction mapping of JVM functionalities. It support as much JVM capabilities as possible for nontrivial Java programming on resource constrained devices. Basic KVM features and its responsibilities can be outlined as follows:

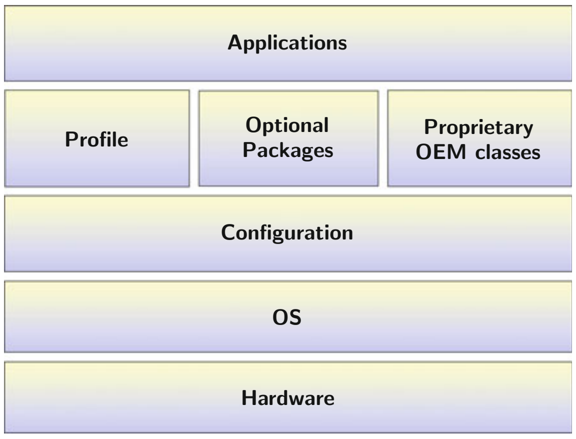


Fig. 8.2 J2ME software stack

- The design goal of KVM is to create the smallest possible complete JVM for running Java programs in an environment limited by a total memory size of few kilo bytes. In other words, KVM is a light weight JVM for hand held devices.
- The static memory foot print of KVM should be under 100 kB.
- KVM should allow programming on the small devices retaining all the important features of Java programming.
- KVM should be portable to any device without requiring retrofitting. However, KVM should be modular, and customizable on the installed devices.
- KVM is written in C. So, it can be compiled and used on both Window and Unix based platforms.

CLDC ensures security in following way. Firstly, all the downloaded Java class files are subjected to a verification step by the virtual machine. Secondly, the applications are executed in isolation under a sandbox environment. Overriding of classes in protected system packages are disallowed.

CLDC also provides a set of APIs for new I/O types and known as Generic Configuration Framework (GCF). GCF is a part of `javax.microedition.io` package. It specifies interfaces for the different types of inputs and outputs. CDC being a superset of CLDC includes GCF. CDC requires GCF for file and datagram support. CDC based profiles requires resource rich devices and does not work on low end devices.

8.2.5.2 Profiles

Several generic profiles are available. Some of these are:

1. Foundation Profile (FP)
2. Personal Basis Profile (PBP)
3. Personal Profile (PP)
4. Personal Digital Assistant Profile (PDAP)
5. Mobile Information Device Profile (MIDP)

Foundation Profile is the Base Profile on which all other profiles are created. FP only targets for the device with network connectivity but without GUI. Personal Basis Profile adds basic user interface to FP profile. The devices which can support complex user interfaces typically use Personal Profile. PDAP defines profile for personal digital assistants. Functionally, PDAP is similar to MIDP. MIDP is responsible for three important tasks: (i) display, (ii) storing of simple database tables in non-volatile memory, and (iii) HTTP based connectivity using CLDC-GFC. MIDP works in an environment where the device is characterized by certain minimum capabilities:

- At least 96×56 pixel display on device,
- At least about 170 kB of non-volatile memory, out of which

1. At least 128 kB should be set aside for running Mobile Information Device (MID),
2. At least 8 kB should be available for storage of MIDlets data, and
3. At least 32 kB should be reserved for JVM operations,

- A keypad, or a keyboard, or a touch screen is available on the device for user's input, and
- The device has bi-directional wireless connectivity.

Apart from hardware capabilities, MIDP also requires following assistance from native OS of the device.

- To be able to run JVM, it is absolutely essential for the host OS to implement exception handling and process interrupts.
- The host OS should provided for the scheduler capabilities.
- Though file system is not a requirement, OS should provide support for basic read and write operations on the persistent storage.

MIDP applications are known as MIDlets in keeping with the well known Java terminology of referring executable program as a (*) lets. MIDlet applications are subclasses of `javax.javamicroedition.midlet.MIDlet` defined in MIDP profile. MIDlets can be viewed as applets on Java enabled mobile phones.

A MIDlet life cycle has three states, namely, (i) start, (ii) pause, and (iii) destroy. In the start state, a MIDlet acquires resources and starts execution. In a pause state, the resources are released and MIDlet remains in the wait state. From the pause state, a MIDlet may either resume in the active state, or enter the destroy state. On entering the latter state, the MIDlet releases all the resources and kills itself by halting the execution of the associated threads. A MIDlets suite is packaged as a JAR file. All MIDlets in a single JAR can access the database of one another. Since, the focus here is Mobile OS, we do not to go into the details of MIDlet programming. An interested reader may refer to other resource including complete reference to J2ME for the same.

8.2.6 Symbian OS

SIBO [19] was an early incarnations of Symbian OS introduced in 1988. It is, essentially, a 16-bit organizer coded in C. SIBO's Unique Selling Point (USP) was power management. It could be ported easily to a variety of devices including PCs. By mid 1990s 32-bit devices started appearing, and SIBO was junked in the favor of a 32-bit system known as EPOC [19]. EPOC was developed with an object oriented approach right from the beginning and was written in C++. Communication capabilities were added, so that hand held devices could access various system services as well as coordinate the accesses to the peripheral devices. It also opened up possibility of expanding communication to provide multimedia services.

Symbian was developed as an open OS support for the devices manufactured by different companies. Nokia, Ericsson, Sony Ericsson together contributed for 75%

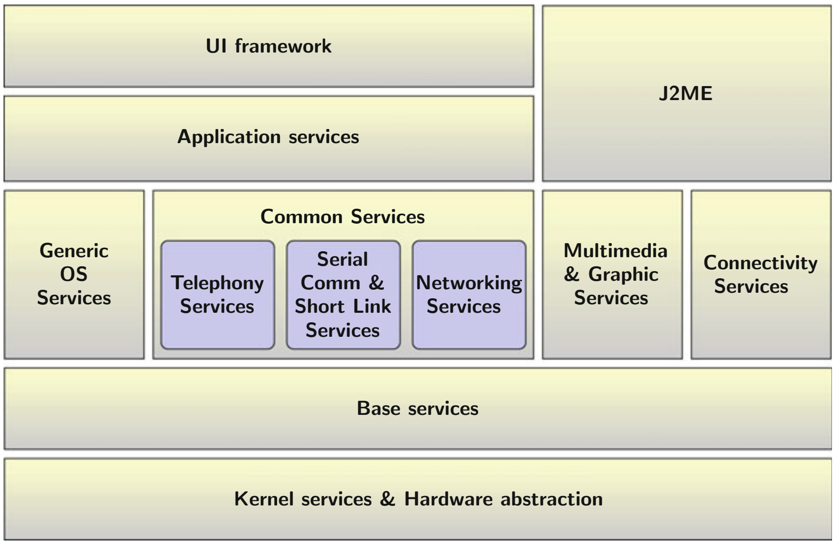


Fig. 8.3 Symbian OS architecture

of the stakes in Symbian. Symbian based phones were immensely successful until Android and iPhone based smartphones started to appear in the market.

Symbian OS ran only on ARM processors. Symbian architecture is described in Fig. 8.3. It separates the user interface from the OS engine. It is a micro kernel based architecture that supports preemptive multitasking and multithreading execution with memory protection. Micro kernel approach reduces kernel size to a minimum by provisioning only essential services. This is appropriate for the resource poor devices. Symbian micro kernel contains a scheduler, provides services for both memory and device management. An important concept for memory protection is data caging. Using data caging an application protects its own data in a private partition which cannot be accessed by other applications. For example, data caging is ideal for the applications which carry out financial transactions.

Accesses to various system services are designed along the lines of the client-server access model. The client can access the services through the APIs exposed by the base service layer. Network, telephony and file system support are accessible through a set of libraries which form the OS base service layer. Networking stacks include TCP/IP (dual mode IPv4/IPv6), WAP, IrDA, Bluetooth, and USB. Most Symbian devices have pre-built telephony supports for circuit switched voice and data (CSD and EDGE ECSD) and packet switched data (GPRS and EDGE EGPRS), CDMA circuit switched voice and data and packet switched data. Furthermore, it is possible to implement newer standards using extensible APIs of the telephony subsystem.

Symbian provides abstraction to the hardware and supports device independent Hardware Abstraction Layer (HAL) which is located below the OS base service layer. Real-time guarantees to both kernel and user threads are provided through kernel. Symbian OS employs a strong security regime. An application is required to present credentials signed by an established security authority for performing any sensitive operation. A Symbian device rely on secured protocols such like HTTPS, TSL, and SSL for management of certificates and encryption. Additionally, the WIM Framework (WAP Identification Module) provides the capabilities to carry out secure transactions using non-repudiation based on digital signatures. Symbian offers a rich set of applications for browsing, messaging, multimedia, over the air data synchronization, file transfers, etc.

8.2.7 Android OS

Android OS was developed by a consortium called Open Handset Alliance led by Google. The first version of android was released in September 2008. There are several versions of android each has a fancy name such as Cupcake, Donut, Eclair, Froyo, GingerBread Honeycomb, IceCreamSandwich, Jelly Bean, Kitkat, Lolipop, Marshallmallow, and Nougat [20]. Nougat is the latest stable release of Android system.

Android OS stack consists of four layers as shown in Fig. 8.4. The bottom-most layer is basically the Linux kernel. This layer should be viewed as the hardware abstraction layer. It performs four important tasks, namely, memory management, interprocess communication, driver support and power management. Driver support is available for a number of drivers including display, camera, keypad, WiFi, flash memory, binder, audio, etc. Through driver support, Android can be ported to new devices. Power management is extremely important for any mobile phone. So, kernel abstractions are useful for enforcing power management polices. Binder driver handles interprocess communication.

The next layer from the bottom consists of a set of libraries including the runtime system for Android. The runtime system consists of a set of core library routines and Dalvik VM. Dalvik VM is essentially an optimized implementation of Java runtime system and provides for a Java like programming framework. Application framework is the next higher layer of the software stack. All the user applications are implemented on the top of this layer.

The life cycle of an Android application is shown in Fig. 8.5. There are three main states in the life cycle of an application, namely, *active*, *pause*, and *destroy*. From the start state, an application migrates to the active state. In active state, the application is in execution. From the active state, an application can transit to the pause state in order to allow execution of another application. An application in the pause state, may either resume its activities or migrate to the destroy state where all its activities are shutdown. An application in the pause state may also be killed, if enough free memory is not available for another application activated by the user. The killed application is reactivated when the user navigates back to the killed application. From the pause state, the application may also be stopped and destroyed or shutdown.

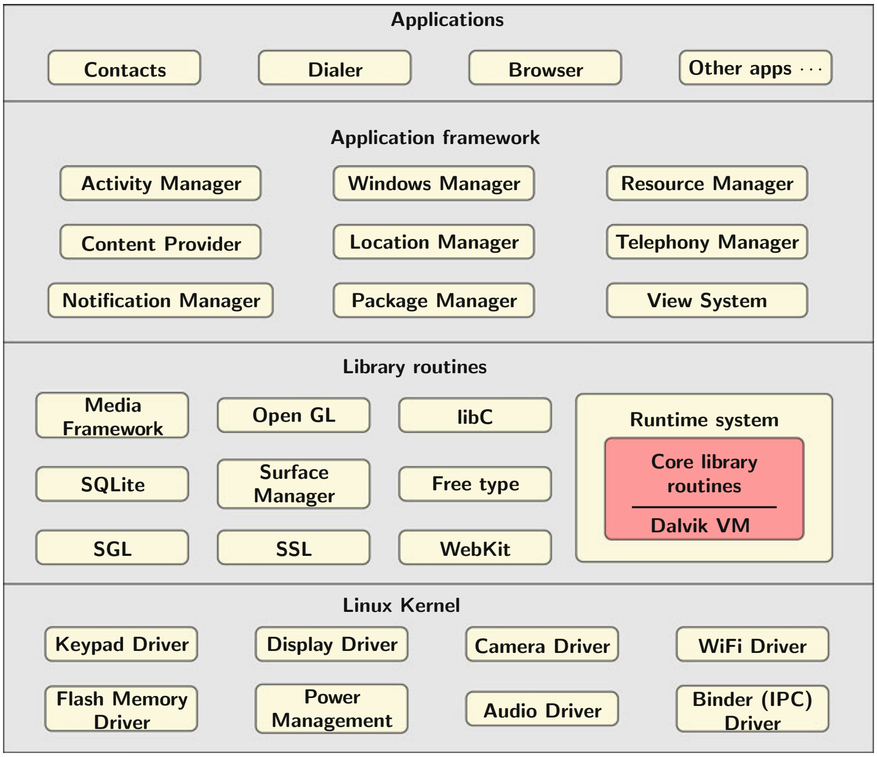


Fig. 8.4 Android software stack

The interesting part of Android memory management is in enforcement of the life cycle model for an application in preference over the usual virtual memory implementation with swapping. At a high level, the application swapping employed by Android and the swapping employed by OS are the same. The aim is to free memory for another active application. Life cycle model avoids the complications associated with the implementation of virtual memory. An important reason for avoiding swapping is to minimize the number of writes on the flash memory. A flash memory has limited longevity as far as the number of writes is concerned. Whereas, repeated writes have little impact on the longevity of a normal hard disk.

Android uses two different types of memory killers to free memory. Out Of Memory Killer (OOMK) is a kill all kind of killer which kills minimum number of applications to free enough memory for a new application. This kill method does not distinguish between applications. So, it may remove important applications from the memory to create free space for a foreground application. On the other hand, Low Memory Killer (LMK) distinguishes the applications in decreasing order of priorities into six different classes, namely,

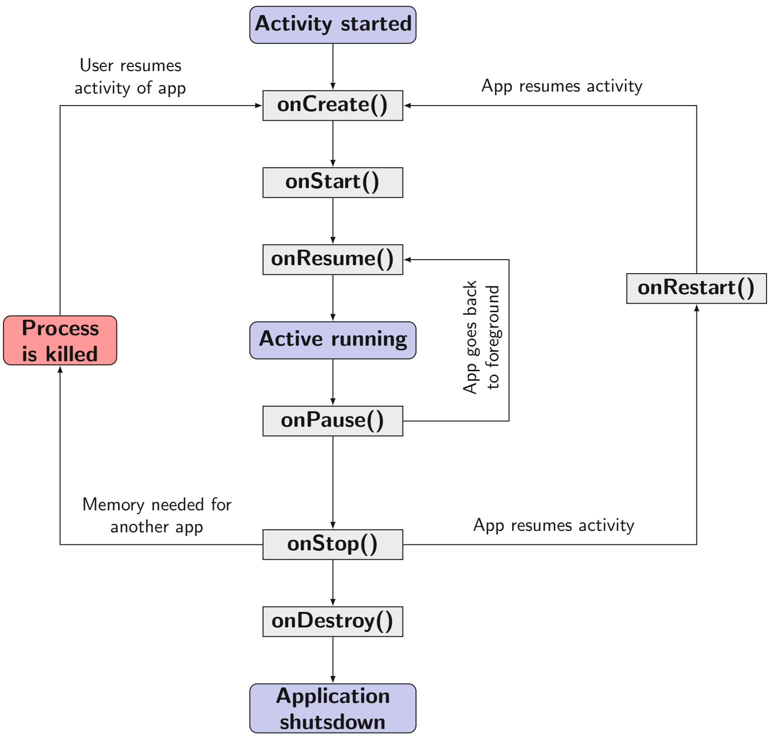


Fig. 8.5 Life cycle of an Android application

- *Foreground_App*: a process which is currently running in the foreground.
- *Visible_App*: a process only hosting the activities that are visible to the user.
- *Secondary_Service*: a process holding a secondary server.
- *Hidden_App*: is a process only hosting the activities that are not visible.
- *Content_Provider*: a process with a content provider.
- *Empty_App*: a process without anything currently running in it.

LMK starts killing applications in increasing order of the priorities until sufficient free space is created in the memory for the foreground application.

8.2.8 Iphone OS (iOS)

Iphone operating system (iOS) like Mac OS X is based on Mach kernel. Mach kernel was developed at CMU in a bootstrapped manner from Free 4.2BSD. It requires about half a GB of storage in the device. Ordinarily, no third party application is allowed. However, jailbreaking can be used to run third party applications in an Apple phone or IPad. The software stack consists of five layers as indicated in Fig. 8.6. Core OS

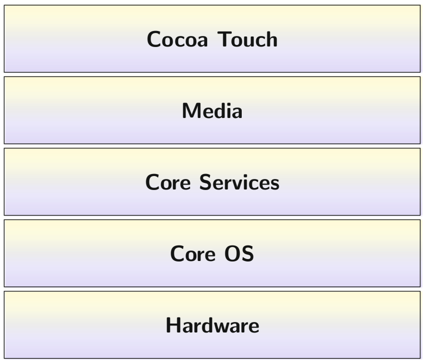


Fig. 8.6 iOS software stack

layer essentially represents the OS kernel. Apple developers performed an evolution of Mach OS X micro kernel (Darwin) make it ready for ARMv5 chipset which was used in iPhones and IPads. Besides kernel, the Core OS layer is made up of drivers and basic OS interfaces. Kernel is responsible for the memory management while drivers provide interface to the hardware. Libraries are used by Apps to access low level features of device hardware. Core service layer basically consists of core foundation network (CFN). CFN presents a set of C based interfaces for the data management and the fundamental software services that are useful to the application environments and the applications. CFN defines abstractions for common data types, and supports internationalization with Unicode string storage. It also provides utilities such as plug-in support, XML property lists, URL resource access, and preferences. The second layer from top is referred to as media layer. Media layer defines a software framework for the application developers to make use of audio, graphics and various multimedia capabilities of an iOS based device in the user Apps. The top layer of the software stack is known as Cocoa Touch Layer (CTL). App developers make use UIKit framework of CTL for look and feel of their Apps. UIkit framework provides support for key apple technologies like multi touch events and control, push notifications, multitasking among others. It provides interfaces to accelerometer, localization and camera. App developers rely heavily on gesture based inputs such as tapping, pinching and swiping which form a part of CTL's touch based technology.

iOS SDK bundles tools necessary for designing, creating, debugging, testing and optimizing softwares. It also includes native and web applications, and dynamic libraries but excludes device drivers. SDK has built-in framework, shared libraries, Xcode tool, iOS simulator and iOS developer's library. Like Eclipse platform, Xcode is used for managing application projects, editing, compiling, running, and debugging code. There is also an Interface Builder tool for assembling the UI. Similarly, an instruments tool is available for performing runtime analysis and debugging. iOS Simulator can be used for testing iOS applications by simulating the iOS technology stack on a Mach OS X. It makes testing faster, since a developer does not need to upload the application to a mobile device. The iOS Developer Library is the source for documentation helpingapplication development. iOS applications are written in

Objective C which is derived from ANSI C and augmented by objective oriented syntax borrowed from Smalltalk.

8.2.9 Comparison of iOS and Android

Symbian foundation has disintegrated completely. The last Symbian based smartphone Purview 808 was introduced by Nokia in February 2012 [21]. From 2012, virtually no Symbian release was publicly announced, as Nokia moved it to closed licensing. Blackberry till date remains a closed platform, and has failed to reach masses. Window mobile has not been quite popular among the general users. Nokia, after abandoning Symbian, has migrated to windows mobile system. According to a statistics [22] Android OS commands 87%, while iPhone commands just about 11% of the market share of mobile phone OS market world wide for the second quarter of 2016. This implies only 2% of the smartphones have other OSes. Therefore, Android and iOS are the two main competitors in mobile OS segments. Our focus in this section is on comparison of these two systems.

Being an open system, Android has several advantages over iOS. Android application can be crowd sourced. The users can get their hands on Android system and customize applications according to their likes and dislikes. However, downloading applications from the sources outside Google play store may introduce viruses or worms in the device. Only Googly play store has a serious vetting process with antivirus features. iOS does not allow Apps to be downloaded from any sources other than the official App store. Apple enforces a code signing process whereby an application developer must sign the code using the Apple issued certificate before uploading the same to the App store. Unsigned codes cannot be uploaded. It ensures that codes do not possess any malwares or viruses. It does not mean apple devices are completely immune to hacking, malware, spyware or viruses. However, the chances of these happening to an iOS device is much less compared to an Android device.

Both iOS and Android implement memory randomization or Address Space Layout Randomization (ASLR) for memory protection. iOS had employed address randomization right from beginning. In Android, ASLR was added since the Jelly Bean release. Memory randomization makes it difficult for the viruses or the malwares to guess the exact locations for attacking the application code.

Encryption mechanism is used in both iOS and Android. But device encryption was not available for Android release lower than 3.0. Encryption API for Android was release first time in Ice Cream Sandwich 4.0. Device encryption in iOS was introduced in iPhone 3GS.

8.2.10 Cross Platform Development Tools

For the application developers, it becomes a productivity linked difficulty if every application needs to be retooled for one or the other OS. Consequently, cross platform development tools have become an important development framework [23] for applications. There are many cross platform tools such as: Rhomobile [24], Titanium [25] and PhoneGap [26]. These tools allow UI development using web technologies like HTML, CSS and Java. However, the UIs developed over native development framework are always rated better than the corresponding UIs developed over cross platform tools.

Some of the important requirements of a cross platform development tool are as follows [23]

- *Multiple OS support*: The platform must support application developments for multiple mobile OSes particularly Android and iOS.
- *Rich UI set*: It should provide support for a rich set of UIs, and possibly include support for customizable parameters based UI programming. Multimedia I/O, 2D and 3D animations enhances the user's experience. So, it should be possible to develop such interfaces on the cross platform tools.
- *Backend connection support*: Many mobile Apps for information, entertainment, navigation, social media interactions need backend connectivity. Therefore, the tools should also provide smooth support for backend connectivity protocols and platform independent data format.
- *Optimization for power consumption*: Energy drainage is a common issue affecting all portable devices, irrespective of native OS support. The generated Apps should be optimized for power consumption.
- *Security and privacy*: Security are privacy are two important area of concerns in use of mobile applications. Typically, mobile phones store highly sensitive and private data concerning personal identity. Therefore, the developers would always be apprehensive about the security support provided by cross platform tools.
- *Installed app extension support*: A extension to old installed Apps should be possible through updates or patches.

A generic layered architecture of a cross platform development tool is provided in Fig. 8.7.

The application developers use web technologies to implement the functionalities of applications. Cross platform tools enable implementation of interface, access of storage facility, and device specific features (e.g., sensors, camera, contacts) which interact with a JavaScript API. The Javascript API interacts with the native API of the mobile platform. Finally, executables for different platforms are generated by building the application.

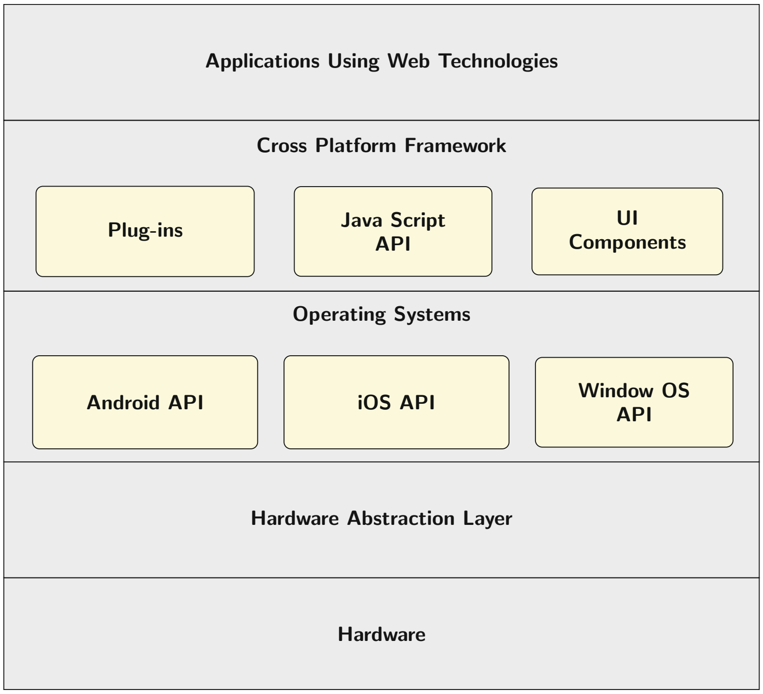
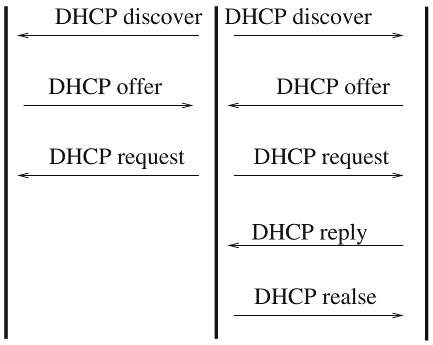


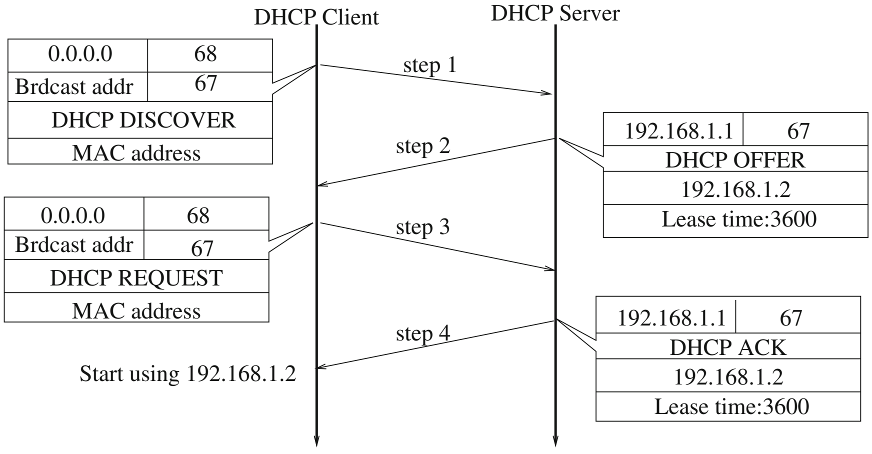
Fig. 8.7 Software stack for a cross platform tool

8.3 Mobile IP

Mobile IP [1] is a modification of wire line IP at the level of Internet enabling mobile users receive messages independent of the point of attachment. A wire line IP address refers to a unique fixed point of attachment for a node. Messages can be delivered by tagging them with IP address and port number pairs. This means an IP address allows to identify a node from millions of available nodes in the Internet. By having a stable IP address assigned to a computer, IP packets from other computers in the Internet can always be delivered at the same computer. In a wired network, the nodes are mostly static with a fixed point of attachment to the Internet. Dynamic Host Control Protocol (DHCP) [27] provides discrete slow mobility to static nodes in a wired LAN. It combines IP address assignment protocol with mobility. Using DHCP, a node can change its point of attachment with limited functionality. Figure 8.8 illustrates how DHCP service operates. For assignment of an IP address, a newly booted machine broadcasts a DHCP DISCOVER packet. A DHCP relay agent is needed for each LAN, as limited broadcast may not reach DHCP server if the server does not belong to the same LAN. DHCP relay agent has the IP address of DHCP server. It intercepts all DHCP packets including DISCOVER packet and send DHCP DISCOVERY packet as a unicast packet to the DHCP server which may be located in a distant network.



(a) DHCP discovery.



(b) DHCP operation.

Fig. 8.8 DHCP operation

DHCP leases out a new IP address to a requesting node for a limited period. The lease period is renewed before expiry as long as the node remains connected and keeps sending renewal requests. When a node changes its attachment to the Internet its IP changes, and as a result all IP clients on the node will stop working. The node needs to restarts its Internet subsystems. Typically, users do not selectively restart the Internet services, but reboot the system. So, the DHCP mechanism, at best, can provide mobility to the extent that the point of attachment of a terminal in the network can be changed. It essentially implies that the requirement of a stable IP address is in direct conflict with node's ability to become mobile.

The portability of nodes enabled through DHCP is also fairly complex. Most application use Fully Qualified Domain Name (FQDN) which should be mapped to an IP address. In order to find mapping of FQDN to an IP address, an application typically seeks assistance from a DNS service. However, when IP addresses are allocated dynamically, the DNS will be unable to resolve the mapping unless mapping updates are provided to DNS for each and every FQDNs. Since DNS is the most important administrative component of a network, any application designed to alter data must have to be tested and trusted fully before deployment. The updates have to be applied to DNS at frequent intervals. So, there is a possibility that the entire network service may go for toss if these updates are incorrect. Specially, in network having large number mobile devices, even providing DNS portability would be fairly complex task.

In view of the technical complications in maintaining IP connectivity of a portable and mobile device to Internet, a simple and easily scalable approach is needed. Notably, the approach should neither affect the protocol stack nor require re-engineering of the applications.

8.3.1 Overview

Mobile IP protocol provides a simple mechanism to deliver packets to a mobile node when it moves out from its home network. Mobile IP has been designed to avoid all the complications of DHCP mentioned above. It supports mobility of nodes through cooperation of three subsystems:

1. A *mobility discovery* mechanism which allows a mobile node to detect its movements in the Internet,
2. An *IP registration* mechanism which allows the mobile node to register its IP address with an agent of the home network after the node obtains an IP address from the new network.
3. A *packet delivery* mechanism which delivers packets to a mobile nodes when it is away from the home network.

The interactions between three subsystems are accomplished through the following three functions:

1. **Agent discovery**. There are routers which advertise their availability to offer service on a wireless link to mobile nodes appearing in the neighbourhood.
2. **Registration**. After a node successfully obtains an IP address (called its care-of address) in a foreign network, registration process enables the node to inform the newly acquired IP address to an agent belonging to home network.
3. **Tunneling**. It is a process by which the home agent encapsulates the datagram meant for a mobile node to be delivered to the care-of address (foreign agent) of the mobile node.

Sometimes an additional function called *reverse tunnel* may also be needed. Reverse tunneling is a process through which the foreign agent encapsulates datagram meant for a correspondent node from a mobile host to be delivered at the home agent. Home agent decapsulates the packet before directing the same to the correspondent node.

In a foreign network, a mobile node effectively uses two IP addresses at a time: (i) a *home address* (HA) and (ii) a *care-of address* (CoA). The HA is static. It is used to identify a mobile node's TCP connections, if any. A CoA is dynamic, which changes as often as the mobile node changes its point of attachment to the Internet while roaming. Therefore, CoA can be viewed as a topologically significant address of a mobile node. CoA specifies the network number that identifies the point of current attachment of a mobile node with respect to the network topology. HA insulates other communicating nodes from the complications of following the mobility a mobile node. Therefore, other nodes on the Internet are oblivious to roaming of a mobile node, and transmit packets to the latter by using its HA.

Mobile IP requires the existence of a topologically static network node or a router called the *home agent*. A mobile node (MN) registers IP of its new point of attachment (i.e., CoA) with a home agent when it is away from home network. The home agent intercepts all the packets having MN as destination, and delivers these packets to MN's current point of attachment in the *foreign network*. So, it is important for the home agent to know a mobile node MN's new attachment point as it moves. MN should register its new *care-of address* (CoA) with the home agent every time its CoA changes. It is the responsibility of the home agent to deliver packets from the home network to CoA of MN. In other words, when a MN is away from home network, HA acts as a proxy for the MN.

All the packets with the MN as destination are intercepted by HA. HA then creates a new packet corresponding to each packet it receives on behalf of MN. It constructs a new IP header with CoA of MN as the destination address. The corresponding old packet forms the payload of a newly created packet. Each newly created packet uses a higher level protocol number indicating that the next protocol header is also an IP header. Then each such a newly created packet is transmitted by the home agent. The packet modification as explained above is essentially a redirection process called *tunneling*. The mobile node's home address is prevented from influencing the routing of the encapsulated packet till it reaches at CoA of the node. The packet from a sender S for a mobile node MN, located topologically in a foreign network, is encapsulated by creating a IP header. The packet is tunneled to foreign agent which decapsulates the packet and redirects it to correct destination (or the mobile node MN).

8.3.2 Agent Discovery

Agent discovery is similar to the one that is used by the nodes in Internet for discovery of routers running ICMP (Internet Control Message Protocol) router discovery protocol [28]. It involves the routers to broadcast router advertisements periodically. The agent advertisement is the most critical component of Mobile IP functions. The structure of the ICMP packet for agent discovery in Mobile IP is illustrated in Fig. 8.9. The packet has following fields:

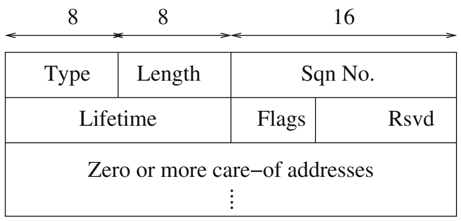


Fig. 8.9 ICMP packet format for agent advertisement

1. *Type*: It distinguishes among various extensions to ICMP router advertisements which may be used by a mobility agent. Provision exists for many extensions, the most important being extension the type 3 which is for the advertisement of mobility agent.
2. *Length*: Specifies the length of a single extension. For example, the length of type 3 extension depends on many care-of address that may be defined. Typically, one care-of address is defined.
3. *Sequence number*: It essentially represents freshness of the router advertisement. Sequence number is incremented by 1 for each successive advertisement.
4. *Lifetime*: Specifies the expiry time of the current advertisement.
5. *Flags*: There are seven flags representing various features supported by mobility agents. We will discuss more about the flags, later in the text.
6. *Care-of address*: IP addresses to be used by mobile nodes.

A home agent need not necessarily offer a care-of address service. But it still needs to broadcast mobility agent advertisement in order to allow a mobile node to detect that it has returned back to home network. A mobility agent need not provide the default router address as it would be found in other ICMP router advertisement.

The flags `H` and `F` respectively represent home and foreign agent. A mobility agent may offer both services, so both `H` and `F` may be set. The flag `B` set to indicate that the foreign agent is busy. However, `B` cannot be set unless `F` is also set. The flag `R` denotes registration required, and is meaningful for care-of address co-located with the mobility agent. Encapsulation protocol preferred by the foreign network is known by three flags: `G`, `M`, and `V`. `M` is used when minimal encapsulation [29] is preferred.

8.3.3 Registration

The process of registration involves a MN registering its CoA with the designated HA in its home network. Figure 8.10 illustrates the sequence diagram of the registration process. As one may notice, it is similar to the process of acquiring IP address from DHCP. Initially, MN sends a registration request to foreign agent. Foreign agent forwards the request to home agent on behalf of MN. Home agent sends a registration reply back to foreign agent which completes the registration process. The format of the registration request appears in Fig. 8.11. Flag `G` or `M` indicates a

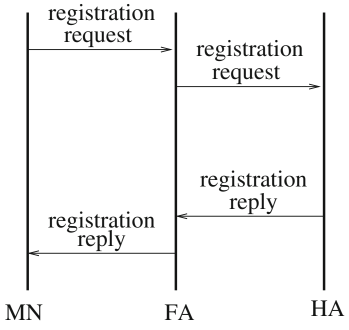


Fig. 8.10 Registration process

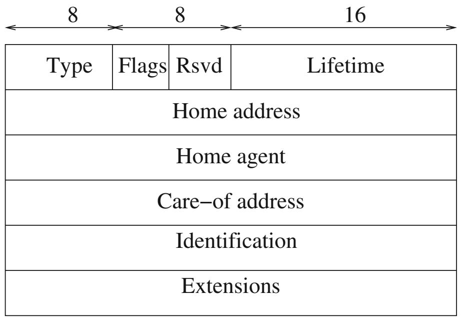


Fig. 8.11 Format of registration request message

home agent whether IP-in-IP or minimal encapsulation is preferred by the foreign agent. Setting `V` bit, in request message, tells the foreign agent that Von Jacobson header [30] compression is desired. Registration message is sent using UDP protocol.

The structure of a registration reply is shown in Fig. 8.12. It consists of type, code, and lifetime field. Lifetime field informs a mobile node about the duration of time for which the registration of the care-of address will be honored by a home agent. When a registration request is rejected, code field tells the mobile node what went wrong. In the case, a registration request is accepted, the code field contains a 0. Request may be rejected either by foreign agent or by home agent. An interesting scenario occurs when the registration request sent by mobile node contains a directed broadcast address for finding a home agent. The registration request is rejected by every home agent. However, the reject replies received by the mobile node contains addresses of the available home agents. So, the mobile node can try again with a valid home agent address for a fresh registration, and it will succeed.

Registration request procedure has been designed to resist two types of attacks: (i) masquerade attack, and (ii) replay attack. In first type of attack, a node impersonates as a foreign agent and diverts all traffic meant for a mobile node to itself by sending a fake registration message. In the second case, a malicious agent replays an old registration message and effectively isolates a mobile node from the network.

In order to guard against malicious users registration, the registration request is protected by inclusion of a non-reproducible value with the identification field of

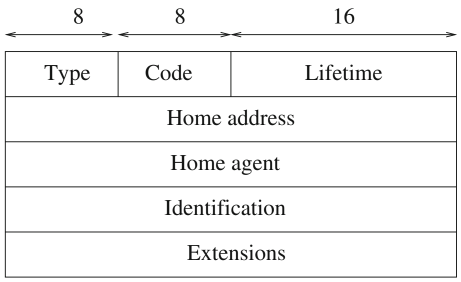


Fig. 8.12 Format of registration reply message

the request. The value changes with each new request. The value may either be a timestamp or a *nonce*. The home agent and the mobile node should agree, before hand, on the unforgeable values that could accompany the registration request.

Three different authentication extensions are defined for use in mobile IP. These are: (i) mobile to home, (ii) mobile to foreign and (iii) foreign to home. Each uses a different security parameter index (SPI). SPI defines the security context to be used to compute authenticator. The context consists of authentication algorithm, and the secret (keys) to be used.

8.3.4 Routing and Tunneling

After the CoA registration for a mobile node becomes successful, the home agent intercepts all datagrams meant for the mobile node. The home agent then tunnels (encapsulates) all these packets to the same to care-of address provided by the mobile node. Although there may be many ways to encapsulate, the simplest one is IP-in-IP [31] encapsulation [32]. Figure 8.13 illustrates the encapsulation technique. The care-of address becomes destination address in the tunnel header. By including IP protocol value of 4, the encapsulated packet indicates that payload carried by it is again an IP packet. The inner packet is not modified except for decrementing TTL value by 1. Compared to IP-in-IP encapsulation minimal encapsulation has less overhead. But it can be used, if all the entities, namely, mobile node, home agent and the foreign agent agree to do so. Minimal encapsulation uses protocol value 55 against 4 of IP-in-IP encapsulation. The length of minimal encapsulation header is 12 or 8 depending on whether or not the original source IP address is present. It allows the original datagram to be fragmented at home agent. However, this encapsulation cannot be used if the original datagram is already fragmented, because there is no room to insert fragmentation information.

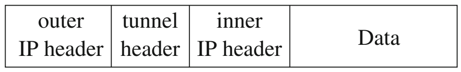
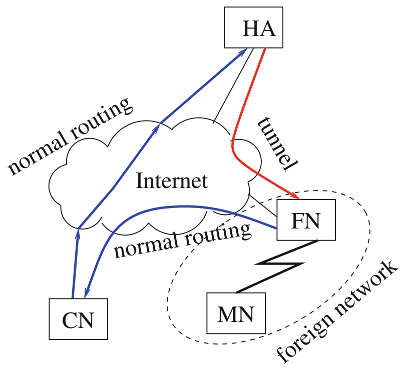


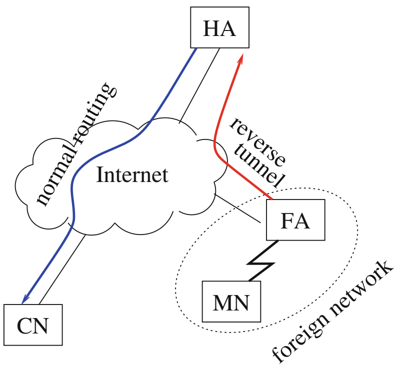
Fig. 8.13 IP-in-IP encapsulation

If the delivery of a tunneled datagram to care-of address fails, then ICMP error message is received by the home agent. But an ICMP error message can incorporate only 8 bytes of datagram in error. So, when an ICMP error message is returned, it may not contain original source of the tunneled packet. Under this scenario, how would a home agent notify about the error to the original source of the datagram? If a home agent keeps track of a care-of address including the sequence number of the tunneled packet then the returned ICMP error message may be matched against the stored information, and a notification can be relayed back to original source. However, there is a small problem in forwarding the notification. The correspondent node must not be aware of the existence of a tunnel. So, a home agent needs to modify ICMP error message from *network unreachable* to *host unreachable*. A home agent also keeps track of many other tunnel parameters such as maximum transmission unit (MTU), TTL for the encapsulated datagrams. The collection of tunnel parameters can be found in [32].

Mobile IP allows a mobile node to send data packets to a host directly using standard IP routing scheme. So, the packet flow in both directions constitute a triangle routing pattern. Inward flow is routed through home agent to care-of address of mobile node, while outward flow from mobile node to a correspondent does not require any rerouting and follows standard IP routing. However, due to security reasons, ingress filtering used at a network may discard packets originating from any foreign node (belonging a different network). This implies that mobile node cannot transmit packets directly to a node outside the network it is visiting. The solution to this problem is to apply a tunneling also in the reverse direction [33]. All datagrams from mobile node in a foreign network can then be routed through the home agent. The tunneling and the reverse tunneling are illustrated pictorially in Fig. 8.14. As it is clear from the figure, with reverse tunneling the overhead of routing data packets between a mobile node and a correspondent node (CN) increases considerably specially when



(a) Tunneling



(b) Reverse tunneling

Fig. 8.14 Tunneling and routing in mobile IP

CN, in turn, belongs to a foreign network. In this case, the two-way traffic between CN and MN has to pass through double-tunneling.

Mobile IP has been designed to operate together with normal IP routing. The network may have both types of nodes: those which implement mobile IP and those which do not. It should be possible for an normal correspondent node, not implementing mobile IP, to communicate with a mobile node. The normal node would use normal IP routing. It sends an ARP (address resolution protocol) request to home network for resolving the address of mobile node before sending any message. This works fine as long as mobile node is actually present in home network. The nodes in the home network may also cache the address of the mobile node for use. However when the mobile nodes moves away to a foreign network, ARP would be unable to resolve any request. Also the nodes that have cached mobile node's address, and attempts to use stale information while trying to communicate with the mobile node. In either case, any attempt to communicate with the mobile node fails.

Mobile IP is expected to handle both the situations stated above. It deals with these situations in the following way. The home agent broadcasts gratuitous ARPs to all nodes. The objective of broadcasting gratituitous ARPs to ensure that all nodes which have cached the MAC address of the mobile node should update the corresponding cache entry. The details of handling ARP request for a roaming mobile node, are as follows:

1. Home agent supplies its own MAC address and continues gratuitous broadcasts for sometime to ensure that every node in the network has received the ARP broadcast and updated their respective cached entry.
2. During the time a mobile node MN is away from its home network, the home agent intercepts any ARP request meant for MN and supplies its own link-layer address
3. As soon as the mobile node MN comes back to home network, the home agent sends another series of gratuitous ARPs to assist other nodes to evict stale cache entries concerning the mobile node.

Mobile IP does not, however, specify a mechanism to update ARP cache in a foreign network. So, mobile nodes should never broadcast an ARP request or an ARP reply packet in a visited network. Otherwise it will be hard to contact mobile node after it moves away.

8.4 Mobile Shell (Mosh)

Before discussing the architectural design of Mosh, let us first understand why a new shell program is necessary at all? SSH [3] has been in use since mid 90s. SSH lacks support for IP mobility. To understand why, consider the environment needed for SSH to run.

1. Firstly, SSH runs on TCP [2]. A TCP connection is represented by a connection tuple: (`src_IP, src_port, dest_IP, dest_port`). A TCP connection breaks if any of the four fields of its connection tuple is modified.

2. Secondly, in SSH every character is echoed and in-line editing is performed at a server or a remote computer.
3. More importantly, in a cellular mobile environment, the performance of a TCP connection is severely affected by micro mobility protocols (handoff) which introduce packet losses and duplications, delays, jitters and break before make, etc. [34].

In summary, the basic assumptions for SSH to operate is an efficient, non-lossy high bandwidth network. It does not understand what to send. Therefore, it sends out everything that an application puts out. The size of data can be in megabytes. To support rendering of SSH output at the client terminal, the round trip time should be typically in few milliseconds. In contrast, in a relatively unloaded 3G network, the round trip latency may exceed several hundreds of milliseconds. The delay may cause the buffers to overflow due to concurrent bulk transfers. A delay of several orders in magnitude of the tolerable range of milliseconds, renders SSH useless for any interactive operations on mobile devices.

8.4.1 Overview of Mosh

Having understood the reasons behind the obvious inadequacies of SSH in dealing with IP mobility, Winstein and Balakrishnan [4] set the design goals of Mobile Shell (Mosh) which offers solutions to the problems mentioned above. Mosh supports IP roaming. It can operate on intermittent and inherently lossy low bandwidth network connection. One of the smart approach employed in Mosh is predictive client-side echoing and line editing. So, Mosh gives the user a feeling that the remote computers are local. This becomes possible because Mosh uses a different approach in local echoing. It runs the terminal emulator at the server-side. The client and the server maintain images of the terminal states synchronized. In contrast, SSH sends an octet stream over the network and hands it over to the terminal emulator on the client side for its display. In summary the two major techniques which makes Mosh advantageous compared to SSH are:

1. *State Synchronization Protocol* (SSP). This is a secure object synching protocol which runs on the top of UDP [2]. SSP synchronizes abstract objects with IP roaming on intermittent and low bandwidth lossy network.
2. *Speculative Display Rendering*. Display is rendered immediately when the client is able to guess with confidence. It is also possible to repair the display even if prediction is not 100% correct.

8.4.2 State Synchronization Protocol

SSH protocol is simple. As illustrated by Fig. 8.15, it consists of two basic elements, namely, a *pseudo terminal* and a *terminal emulator*. SSH client runs the terminal emulator on the top of the pseudo terminal. The server side application conveys the output octet stream over encrypted TCP connection that exists between the client and the server. The pseudo terminal on the SSH client uses the terminal emulator for rendering the display on screen.

The technical contribution from Mosh research [4] was to synchronize abstract objects instead of octet stream. The goal was to build a low latency object synchronization protocol and to combine this with a local user interface that replaces SSH. The protocol does not need to know the inner details of an object, but it can distinguish between two states of an object. An object also has to provide a *patch vector* that can be applied to its previous state for synchronization of the two states. It is the object implementation that determines the synchronization. In other words, if the object implementation is incorrect, then SSP may lead to wrong synchronization.

In the specific case of mobile terminal emulation, the recent most state of the screen is sent to a client by the server using a frame rate that adjusts to the network conditions. So, the network buffers do not experience overflow conditions. The server side can buffer the generated keystrokes and optimize the transfer of the resulting octet stream. However, this luxury is not possible on the client side which must send each and every keystroke. SSP runs in both directions, client and server. At the client side, the objects represent the history of the user's input while at the server side the objects represent the responses in terminal window.

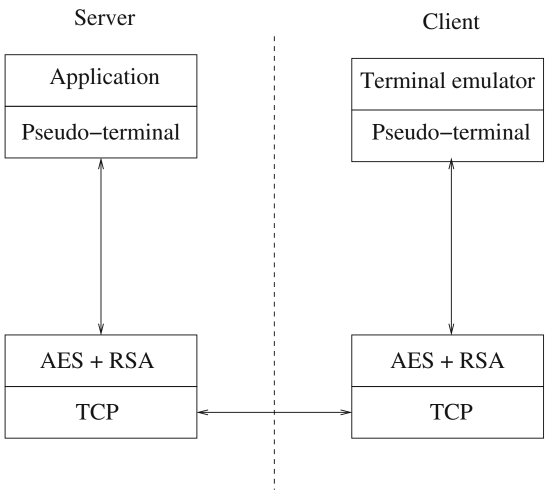


Fig. 8.15 Simplified view of SSH protocol [4]

The design goals of SSP are as follows:

1. It should ensure authentication and login through existing infrastructure, namely, SSH.
2. It should not use any privileged code.
3. It should update the remote host to the sender's current state as fast as possible.
4. It should allow a roaming client to change its IP, and the change should be transparent to the client.
5. It should recover from the errors due to dropped or reordered packets.
6. It should ensure authenticity and confidentiality.

Since SSP does not have any privileged code, the key exchanges take place outside the main SSP activities. The bootstrapping of the terminal session performed by running a script under a privileged server. Login to a remote host is performed through established means of a connection like SSH. The server program listens on a higher order UDP port and provides a random shared encryption key. After this the SSH connection is no longer needed, and can be stripped down. This approach represents a smart way of leveraging existing infrastructure for a reliable and secure authentication. After login becomes successful, and SSH connection has been teared down, the client talks directly on the established UDP port.

As shown in Fig. 8.16, SSP basically works on object layers and runs on UDP. The server side exports an object called screen. This object has the contents of the terminal display from application running at the server. The client side, on the other hand, exports the object called key strokes which is a verbatim transcript of the key strokes made by the user of the client.

Datagram layer maintains roaming connectivity. The payload received from the transport layer is appended with an incrementing sequence number, and then encrypted by the datagram layer before being sent it as a UPD datagram. The

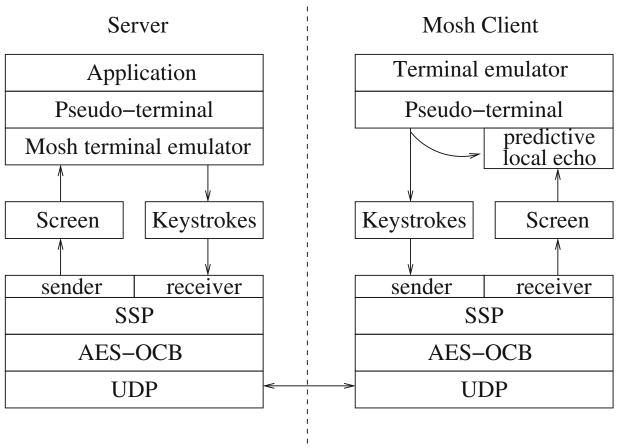


Fig. 8.16 Design of Mosh protocol [4]

datagram layer is also responsible for maintaining the round trip time estimate of the link and the user's public IP address.

8.4.2.1 Authentication and Privacy

Mosh encryption mechanism is designed on the top of AES-128 [35] in Offset Code Book (OCB) mode [36]. OCB integrates message authentication with block cipher encryption. Therefore, it provides both privacy and authenticity through only a single key. Repeated and reordered packets are handled by `diff` operator. Every datagram represents a `diff` between a numbered source state and a numbered target state. No replay cache or history state needs to be maintained.

8.4.2.2 Mobility

The server comes to know about client's roaming by naming connection independent of network. The connection which is traditionally known by IP and port numbers, now mapped to a sequence number protected by encryption. When the server receives an authenticated packet from the client with a higher sequence number, it stores client's IP and UDP port numbers from the received datagram. So knowing about client's roaming becomes simple and automatically maintained. The approach appears bit of a reversal of SSH where application writes into a TCP socket and oblivious to network conditions. However, SSP forces the application to be aware of the network conditions and the alerts. SSP is responsible for the rest of the synchronization for maintaining the connection.

8.4.2.3 Speculative Local Echo

Another major function of datagram layer is speculative local echo. The client side terminal emulator makes a prediction on the effect of keystrokes made by the user and later verifies the same against authorative screen sent by the server. The user's keystrokes in most of the Unix applications are echoed at the current cursor location. So, it is possible to approximate the response from the remote application.

Transport layer synchronizes the states of the local and the remote hosts. The transport sender updates the receiver to the current state by sending three things, namely, the state of the sender, the state of the receiver and the binary difference between two states. The difference is logical difference, and the exact details are computed from the object type and object implementation and do not depend on SSP. For instance, the user's input depend on every keystroke whereas the difference in screen states is defined by creating a minimal message that changes the client's screen to the server's current screen.

8.4.2.4 Update Frame Rate

As SSP does not need to send every octet it receives from the remote host, the frame rate can be controlled to suite the conditions of the network. The minimal interval between frames is set to half the RTT estimate. So, just one instruction will ever be in transit from the client to the remote host. The maximum frame rate is fixed at 50 Hz which roughly matches the limit of human perception. This frame rate also saves unnecessary traffic on low-latency links.

The transport sender uses delayed acks to reduce the volume of the packets. A delay of 100 ms was found to be sufficient to piggyback a delayed ack on the host data. The server also takes a pause since the time its object is modified before sending off an instruction. This is a clever way of avoiding sending of partial updates in quick successions. A collection interval of 8 ms was recommended as optimal on the basis of some application traces.

SSP also sends heartbeats, so that the server gets fast updates on the change in client's IP. Additionally, it helps the client terminal to warn the user when there was no recent response from the server. An interval of 3 s was used for sending a heartbeat. It reduces unnecessary traffic due to heartbeats but at the same time found to be sufficient for handling IP roaming.

8.4.3 Design Considerations of Terminal Emulator

Terminal emulator has to be redesigned suitably in order to comply with the requirements of the SSP's object interface. The client's keystrokes are sent to the server and applied on the server side emulator. The authoritative state is maintained by the server side terminal and SSP synchronizes this back to the client. The client speculates the effects of keystrokes, and in most cases applies them immediately. The client learns about the success of its predictions when receives the authoritative state from the server. This observation allows it to raise its confidence level in deciding whether to display the predictions on the client terminal. In case of high latency connection, the predictions are underlined to ensure that user is not misled about the result. As and when the server's responses arrive the underlined text disappears gradually. If there are mistakes in predictions, they are removed within one RTT time. This approach of predictive display works because most Unix applications either echo the user's keystroke at the current cursor location or wait for the response from the application. In fact, the choice is between display of keystrokes or to wait for the octet streams to arrive from the server. In high latency networks, clearly the choice for speculative echo boosts interactiveness and distinguishes it from one with packet losses. The predictions are not made immediately, the strategy used is to predict when the user hits a key but not to display it immediately. The predictions are grouped together in form of an *epoch*, and displayed together. So either all predictions for an epoch are correct or incorrect. The predictions are made in the background. If one prediction is confirmed by the server then all predictions for the epoch are displayed immediately to the user.

The prediction of echo and its display needs assistance from the server side. In order the display to function correctly, the Mosh client should be able to get a confirmation on its prediction from the server. The initial attempt was to determine whether the predicted echo was displayed on the screen by the time Mosh server has acknowledged the keystroke. This strategy was not satisfactory, because some applications could take a long time to respond after the input was presented to them. So, the server's acknowledgement for a keystroke may arrive before echo is present in the screen state. This leads the client to conclude that the prediction is wrong even if the response from application was correctly predicted. It produces flickers on the screen as the echo is initially displayed, then removed, and when the response arrives from the server, it is displayed again.

One solution to remove flickers was to use a timeout on the client side. A prediction can be considered as correct until a certain timeout occurs or the server's response arrives. However, due to network jitters the acknowledgement from the server may arrive well after the timeout. Using a long timeout interval does not make sense as there is a possibility that the incorrect prediction would stay on for a long time.

The solution that works out well is to implement a server side timeout. This timeout is fixed at 50 ms. This value is arrived at on the consideration that the timeout interval should be fast enough to detect a mis-prediction and at the same time should be sufficiently long for a response to arrive from a reasonably loaded server. So, the client is associated with a terminal object. It contains an echo ack field that represents the latest keystroke given as input to the application for at least 50 ms, and the effect of the keystroke should appear on the current screen. This ensures that client does not timeout on its own. So network jitters cannot affect the client ability to predict echos correctly. The down side is that the server keeps sending an extra packet every 50 ms to send an echo ack after a keystroke.

8.4.4 Evaluation of Mosh

As the paper indicates [4], the Mosh was evaluated by using traces from 40 h of real-world usage by six users who contributed about 10,000 keystrokes together. The plot reproduced from original paper shown in Fig. 8.17 indicates that 70% of the time Mosh was confident in prediction and the responses were almost instantaneous. A major portion of the remaining 30% keystrokes where Mosh predictions failed were for navigations such as north, south east or west of an application screen such as the mail client. So latency distribution in these cases are as comparable as SSH. When Mosh prediction was erroneous, it was able to correct the keystrokes within RTT time for 0.9% of keystrokes. The errors were found to occur due to word wrap, i.e., for the characters which were printed near the end of a line moved to the next line at an unpredictable time. Overall Mosh response latency was less than 5 ms compared to 503 ms for SSH. Mosh is available for download from http://www.mit.edu. Elaborate

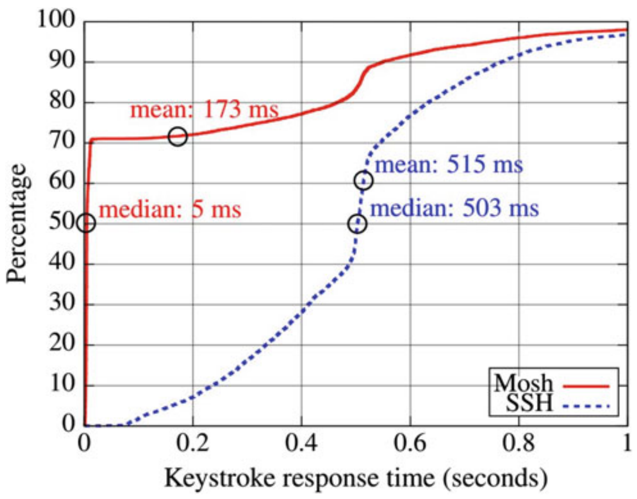


Fig. 8.17 Comparative evaluation: SSH versus Mosh [4]

installation instructions for all flavors of Linux environments, Android app for client terminal and also Chrome add-on are available for Mosh terminal.

8.5 Wireless Application Protocol

Wireless Application Protocol (WAP) [37] defines a set of standards for developing mobile services over wireless networks. The necessity of defining protocol standards for mobile services arose due to the facts that

1. Internet protocols are neither designed to operate efficiently over mobile networks nor optimized for low resources, and
2. Small factor of mobile devices. For example, it is not possible to fully display desktop versions of web sites.

At first we need to understand why a new protocol such as WAP is necessary at all when HTTP (Hyper Text Transfer Protocol) together with HTML have been used so successfully for web services over wired networks.

8.5.1 Performance Bottleneck Faced by HTTP

HTTP defines the information exchange standards over world wide web [38]. It supports information flow between a web browser (HTTP client) and a web server (HTTP server). It is a stateless, request-response type of protocol designed for networks with high bandwidth and long latencies. Though HTTP is independent of transport protocol, a TCP connection is implicitly assumed for lossless information exchange.

Being dependent on the establishment of a TCP connection, HTTP suffers all TCP related performance inadequacies. Some of the notable reasons for performance degradation due to TCP are:

- For establishing a TCP connection a three way handshake is necessary.
- The lower bound of transaction time for a client's HTTP request to reach the webserver is at least two RTTs.
- The default size of a TCP segment is 536B for remote connection though TCP can negotiate for a maximum segment size of 1460B. So TCP breaks the output stream into smaller chunks before sending.
- The performance of HTTP is also affected due to TCP slow start mechanism.

Apart from the TCP related performance bottleneck, there are other problems with HTTP such as:

- It uses long protocol headers, transmits the contents in uncompressed form, and employs primitive caching mechanism.
- It expects two communicating ends to speak HTML [39] language.
- Though multiple HTTP requests can be made over a single TCP connection, each object request has to be a separate transaction. This implies that every request has to make a separate trip.

HTML is basically a description language for passing the data through HTTP. HTML assumes high resolution, high color displays, existence of mouse and hard disk. Typically, the web page designers ignore capabilities of end system, they only focus on optimizing the design of the webpage. In summary, HTTP is designed for networks with high bandwidth and long latencies.

The overall picture of Internet communication model over HTTP is illustrated by Fig. 8.18.

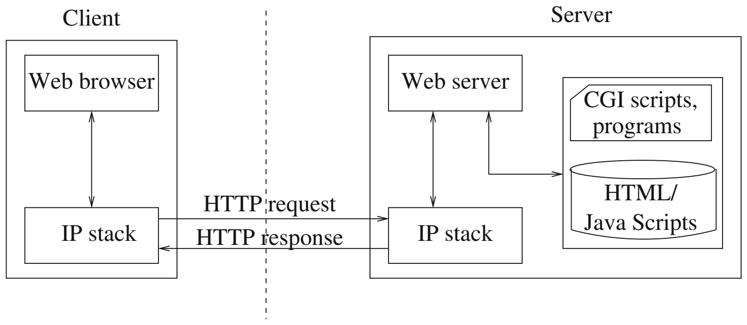


Fig. 8.18 World wide web communication model

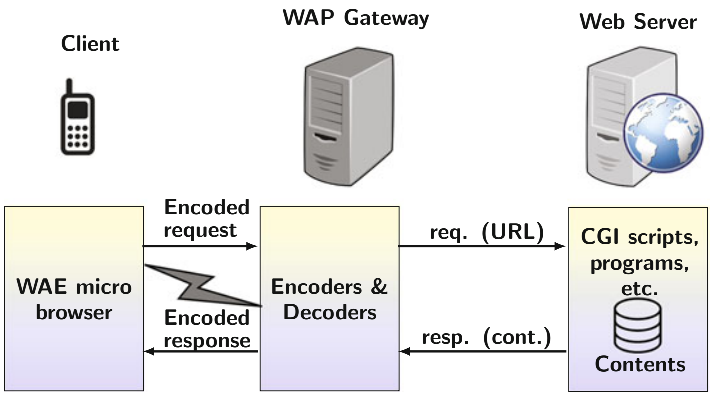


Fig. 8.19 WAP programming model

Cellular service providers anticipated enormous scope for business reach that could lead to effective globalization by having real time interactive services related to travel, banking, finance, weather, stock trading, etc. However, cellular service providers quickly realized that in view of the limitations of HTTP/HTML mentioned above, the existing web infrastructure has to be re-engineered for defining a new communication standard over cellular network. This led to Ericsson, Motorola, Nokia and others to form a consortium for working on a standard which will enable mobile application developers to build interactive applications with enhanced interactive value added services that provide near real-time responses.

As shown in Fig. 8.19, WAP programming model consists of three main components: the client, the WAP gateway and the web server. The important elements of WAP programming model are:

1. *WAP device*: It is basically a wireless enabled, portable client device such as a smart phone, or a PDA.
2. *WAP client*: An entity which receives the web contents from the WAP gateway.
3. *WAP content/application server*: An entity where information or web/WAP application resides.
4. *WAP gateway*: Also known as WAP proxy. It acts both as a client and a server and performs translation of WAP protocol stack to WWW protocol stack. It also has content encoder/decoder.

The client's micro-browser uses web server contents accessible over the network. In a wired network, a HTTP `GET` request is made by the client to the web server. The content of the URL in the form of a HTML page sent as a HTTP response to the client. However, services created through HTML page are not only excessively resource intensive but also too heavy for mobile devices, considering display limitation of small screens. Compute-intensive services carrying redundant information should be optimized for their accessibility on mobile devices. WML (Wireless Markup Language) standards [40] were developed precisely for this. Typically WML pages are encoded in binary in order to reduce the amount of data transmission over a wireless interface. WAP architecture provisions a WAP gateway for such encoding of the WML pages. The gateway machine acts as a mediator between wired and wireless network domains by helping in protocol conversion and data formatting. A WAP gateway performs following four important tasks as indicated by Algorithm 2. The HTTP request, created by the WAP gateway at line 1, is processed for the URL by an appropriate web server which delivers either a static WML file or processes a CGI script to return a dynamically created WML file. The WML file is wrapped by a HTTP header and sent back to the WAP gateway. The received WML is compiled in binary format at line 5, before being sent back as a response to the request of the mobile phone at line 6. The phone's micro-browser processes the WML file received from the WAP gateway to display the contents on the screen.

Algorithm 2: Actions of WAP gateway

```
  begin
      // Request is binary encoded
1     receive a WAP-request from a mobile device;
2     translates WAP-request into a HTTP form for URL;
3     sends HTTP-request;
4     await for a WML response from an web server;
5     compile and encode received-WML in binary format;
6     send encoded WML response back to mobile;
  end
```

8.5.2 *WAP Protocol Stack*

The protocol stack for WAP is shown in Fig. 8.20. It consists of several components which include wireless application environment (WAE), session (WSP) and trans-

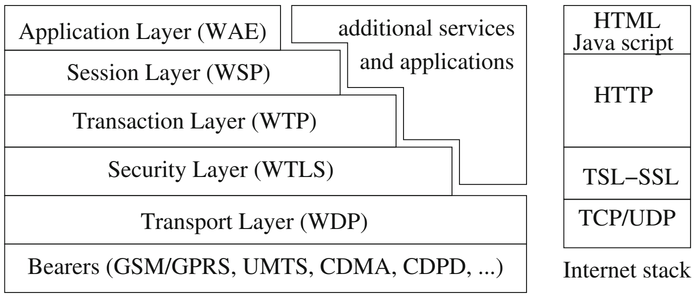


Fig. 8.20 WAP protocol stack

action support (WTP), security (WTL) and data (WDP). At the bottom of the WAP stack, WDP sits on a number of wireless service bearers such as GSM, SMS, CDMA, HSCD, CDPD, HSCD, etc. With respect to OSI reference model, WAE is analogous to Application Layer. WSP, WTP, WTL, and WDP have responsibilities comparable to corresponding layers of OSI reference model as illustrated by Fig. 8.20.

8.5.2.1 Wireless Application Environment

WAE provides environment for a mobile client to interact with web applications through a micro-browser. So the basic elements of WAE are as follows:

- Specification for a micro-browser that controls the user's interface and is able to interpret both WML and WML script.
- Content generators applications, written in WML script, which produces the content formats in response to a specific micro-browser requests.
- Content encoders, such as wireless bitmap, which allow the micro-browser to support color, images, audio, video, phone book, animation, etc.
- Wireless telephony application (WTAI) is basically a collection of extensions for call and feature control mechanisms.

Micro-browser acts as the user agent, and initiates the requests for contents. Apart from the requested contents, additional contents can be generated depending on the capabilities of the mobile device. The device capabilities are inferred from the user agent based on standard negotiation mechanism. This negotiation is performed through WSP mechanism. Three important characteristics exchanged during the negotiation are: versions of WML, WML-script supported, support for image format, and floating-point.

Wirelss Markup Language:

WML is an XML based markup language, designed specifically for wireless handheld devices, optimized for small screens and limited memory capacity. A WML document is called a *deck* and a WML page is called a *card* which is a separate viewable entity and the basic unit of navigation like an HTML page. Sometimes, people try to compare WML entities to HTML entities. But there are a few difference. Each viewable HTML page is represented by a separate HTML file. But a WML deck is represented by a URL address like an HTML page. WML files are stored as static text files on a web server. These files are encoded in binary format by the WAP gateway for onward transmission to a browser by the wireless connection. HTML page undergo no such transformation when transmitted to client. WML browser reads an entire deck of cards at a time. So, while navigating between the cards, the browser does not need any interaction with the web server. This structure is organized by keeping in mind that typically phone users quickly flip through cards before viewing a specific card of their interest. While creating code for a card, WML developer should be aware of the screen boundaries of a phone display. One of the important problem that one has to worry about is that there are many WAP devices of varying screen

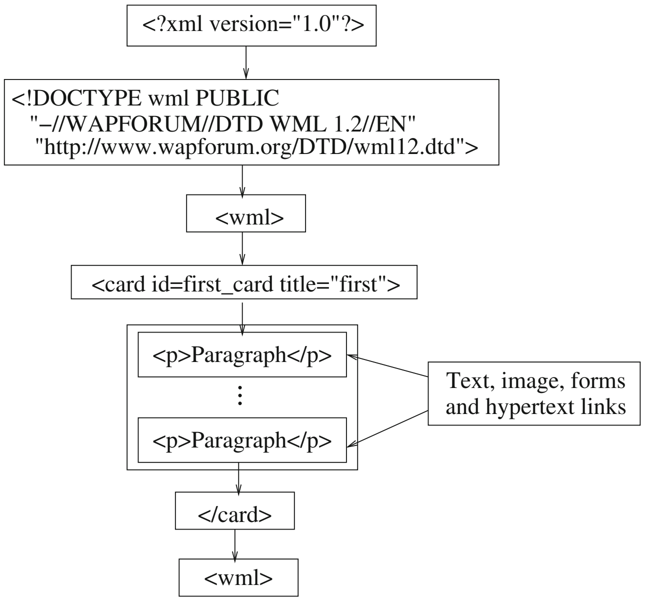

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "–//WAPFORUM//DTD WML 1.2//EN" "http://www.wapforum.org/DTD/wml12.dtd">
<wml>
<card id=first_card title="first">
<p>Paragraph</p>
<p>Paragraph</p>
Text, image, forms and hypertext links
</card>
<wml>

Fig. 8.21 Structure of a WML deck

sizes. Therefore, WML card size needs to be tested at least on a set of few important phones. On the other hand, an HTML developer does not have to worry about screen boundaries, as static devices have sufficiently large screens.

The structure of a WML document or deck is shown in Fig. 8.21. First two lines declare that the document is an XML document and its document type definition (DTD) [41] with its URL. The first line specifies the version number of XML being used. A deck can have many cards, the figure shows only one card. The structure of other cards will be similar. The card may have links to other WML documents, images, and media files. Figure 8.22 illustrates an example of a WML document. Figure 8.22(a) gives the WML file for the page which is displayed in Fig. 8.22(b).

WMLScript:

As explained earlier, WML was created to provide web service accessibility similar to HTML on mobile devices in a resource optimized manner. Dynamic code cannot be written using WML. WMLScript, developed precisely for this purpose, is based on ECMAScript which is a standardized version of JavaScript [42]. However, WMLScripts are not embedded within WML pages, WML pages contains URL links for WMLScripts. So, the WMLScript code is placed in a file separate from WML. The scripts are compiled at the server into byte code before being sent back to clients. They serve the following important functions:

1. Validation of the user's inputs.
2. Accessing functionalities of a user agent.
3. Generation of messages and dialog boxes for alerts, errors.

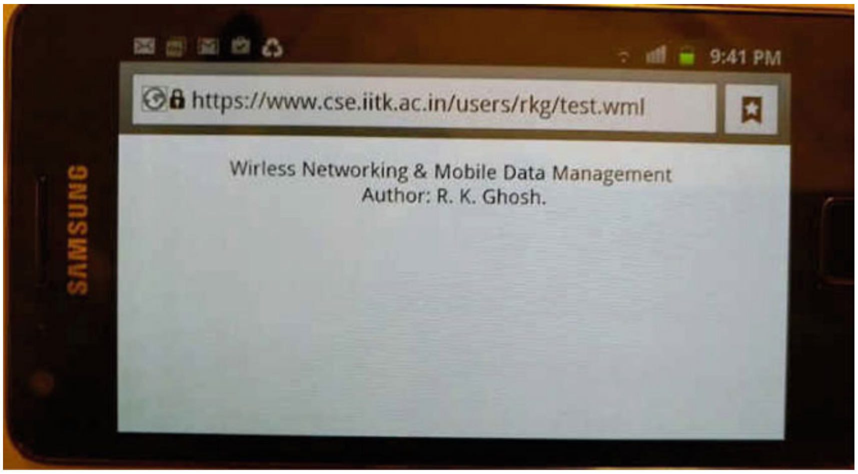
```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">

<wml>

<card title="Card Example">
<p align="center">
Wirless Networking &amp; Mobile Data Management <br /> Author:  R. K. Ghosh.
</p>
</card>

</wml>
```

(a) Source WML document.



Wirless Networking & Mobile Data Management
Author: R. K. Ghosh.

(b) Micro-browser display.

Fig. 8.22 Example of a WML deck

4. Seeking confirmation from the user for execution of certain actions. It basically saves a round trip time and bandwidth for giving a response.
5. Extension of WAP devices for accessing device or vendor specific APIs.
6. Addition of conditional logic by downloading procedural logic as required.

Many useful functions are available in standard WMLScript library. The details about using WMLScript library is beyond the scope of this book. However, for the sake of completeness, we just include a small example in Fig. 8.23.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN" "http://www.wapforum.org
/DTD/wml13.dtd">

<wml>
  <card id="card1" title="Using WMLScript">
    <p>
      <a href="bookScript.wmls#bookTitle()">Run WMLScript</a><br/>
      $(message)
    </p>
  </card>
</wml>
```

(a) Source WML document.

```
extern function bookTitle()
{
  WMLBrowser.setVar("message", "Wireless Networking &amp; Mobile Data Management<br>
R. K. Ghosh")
  WMLBrowser.refresh();
}
```

(b) WML script for the deck in Figure 8.22(a).

Fig. 8.23 An equivalent WMLScript for a WML deck

8.5.2.2 Wireless Session Protocol

WSP (Wireless Session Protocol) provides connection based services to the application layer. It creates an active session between the client and the WAP gateway in order to facilitate the content transfers. WSP also enables thin mobile clients to operate efficiently over low bandwidth cellular and wireless communication networks. WSP browsing application is based on HTTP 1.1 standards [38] with some additional features such as mobility of client terminal, binary encoding, push data functionality, service parameter negotiation, session suspend and resume. WSP supports both WDP and WTP. WDP (Wire Datagram Protocol) is a connectionless datagram service, whereas WTP (Wireless Transaction Protocol) is a connection oriented service defined on the top WDP.

8.5.2.3 Wireless Transaction Protocol

WTP operates on the top of unreliable datagram service. Therefore, WTP is built as a transaction or request-response model well suited for the requests for web contents. WTP cannot handle streams. So, TCP based streaming applications such as telnet or SSH cannot be carried over WTP. However, WTP provides reliability on the top of unreliable WDP.

8.5.2.4 Wireless Transport Layer Security

The most important security weakness in WAP can be attributed to use of the WAP gateway. WAP Transport Layer Security (WTLS) used between the WAP gateway and the device. WAP does not specify any security infrastructure for exchanges between a WAP gateway and a web server. After information has been delivered at the WAP gateway, it flows in *clear text* unless a SSL session is established between the WAP gateway and the web server. Even if SSL/TSL is used, the WAP gateway has access to unencrypted data for a brief period of time before it flows out to the web server. So it is the responsibility of the gateway vendors to ensure that no unencrypted data gets stored into the disk, and the memory used by decryption and encryption processes are cleared before the same is handed over to the OS of the machine. WAP gateway should prevent OS from swapping in or swapping out member pages. WAP standards have no explicit specification for the trust that is expected to exist between the mobile device user and the WAP gateway. So, for any sensitive data such as electronic banking transactions, a bank should not rely on a client's default WAP gateway [43].

WTLS security protocol operates above transport layer and is based on TLS. TLS, built over Secure Socket Layer (SSL) which is a widely used security protocol for the Internet applications such as email, online payment systems, etc. WTLS's responsibilities are to ensures data integrity, data confidentiality, authentication and protection against DoS (Denial of Services) attacks. However, the responsibility of WTLS ceases once the communication reaches WAP gateway where wireless network terminates. In a wired environment, a web client and a web server communicate directly. Therefore, end-to-end security could be ensured by SSL session. Over wireless when transaction is made on a mobile phone (client), WTLS first establishes a session between a WAP gateway (server). The security parameters such as encryption protocols, public keys, and other associated long term security associations are negotiated during this handshaking phase. There is also a lightweight handshaking phase where security parameters are not negotiated but these are borrowed from another session. After the session has been established, all communication between client and server are encrypted accordingly. It is possible for a session to last for few days. However, allowing a session to continue for long increases probabilities of attack. So, WTLS allows keys to be also renegotiated during a session. Though WTLS allows use of certificates, it demands storage at mobile nodes and incurs cost for transmission over the wireless interface.

WAP also provisions for a tamper-proof device called Wireless Identity Module (WIM). WIM is an WAP equivalent of SIM in GSM. It provides private and public keys to perform digital signature and verification of security certificates.

8.5.2.5 Wireless Datagram Protocol

WDP (Wireless Datagram Protocol) is the basic elements of the WAP transport layer. It specifies how the existing data transport services should be used to deliver data to the upper layers. WDP is an adaptive datagram service, and supports different bearer services. For example, if IP bearer is used then WDP protocol adapts UDP, while for SMS bearer it specifies mandatory use of source and destination port numbers. All the details are hidden from the upper layer which see WDP as the only consistent vehicle for data transport. Therefore, WTLS can operate in a consistent manner over WDP.

References

1. C.E. Perkins, in *Mobile IP Design Principles and Practice* (Pearson Education, 1998)
2. J. Postel, Transmission control protocol, https://tools.ietf.org/html/rfc793. September 1981 (RFC 793)
3. T. Ylonen, C. Lonvick, The secure shell (SSH) connection protocol, https://tools.ietf.org/html/rfc4254. January 2006 (RFC-4254)
4. K. Winstein, H. Balakrishnan, Mosh: an interactive remote shell for mobile clients, in *2012 USENIX Annual Technical Conference* (Sunny Boston, Mass., USA) pp. 171–182, 13–15 June 2012
5. O.M. Alliance, Wireless application protocol architecture specification, http://technical.openmobilealliance.org/affiliates/wap/wap-210-waparch-20010712-a.pdf. 12 July 2001
6. The Statista Portal. Number of smartphone users worldwide from 2014 to 2020 (in billions), https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. Accessed 27 Dec 2016
7. G. Martin, H. Chang, System-on-chip design, in *ASICON 2001, 4th International Conference on ASIC Proceedings* (2001), pp. 12–17
8. F. Engel, I. Kuz, S.M. Petters, S. Ruocco, Operating systems on SOCS: a good idea? in *25th IEEE International Real-Time Systems Symposium (RTSS 2004)*, pp. 5–8
9. H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, D. Estrin, Diversity in smartphone usage, in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services* (ACM, 2010), pp. 179–194
10. C. Shin, J.-H. Hong, A.K. Dey, Understanding and prediction of mobile application usage for smart phones. in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (ACM, 2012), pp. 173–182
11. T.M.T. Do, J. Blom, D. Gatica-Perez, Smartphone usage in the wild: a large-scale analysis of applications and context, in *Proceedings of the 13th International Conference on Multimodal Interfaces* (ACM, 2011), pp. 353–360
12. A. Rahmati, L. Zhong, Studying smartphone usage: lessons from a four-month field study. IEEE Trans. Mob. Comput. **12**(7), 1417–1427 (2013)
13. The eLinux Community Portal. Android kernel features (2015), http://elinux.org/Android_Kernel_Features. Accessed 28 Dec 2016
14. F. Maker, Y.-H. Chan, A survey on android vs. linux. Technical report, University of California, 2009
15. H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, D. Boneh, On the effectiveness of address-space randomization, in *Proceedings of the 11th ACM Conference on Computer and Communications Security* (ACM, 2004), pp. 298–307
16. H. Bojinov, D. Boneh, R. Cannings, I. Malchev, Address space randomization for mobile devices, in *Proceedings of the Fourth ACM Conference on Wireless Network Security* (ACM, 2011), WiSec'11, pp. 127–138
17. H. Lockheimer, Android and security, google mobile blog (2012), http://googlemobile.blogspot.fr/2012/02/android-and-security.html. Accessed 27 Dec 2016
18. Neal Leavitt, Mobile phones: the next frontier for hackers? IEEE Comput. **38**(4), 20–23 (2005)

19. B. Morris, in *The Symbian OS Architecture Sourcebook: Design and Solution of a Mobile Phone OS* (John Wiley & Sons, 2007)
20. Community Editors, Android version history (2016), https://en.wikipedia.org/wiki/Android_version_history. Accessed 28 Dec 2016
21. I. Lunden, Nokia confirms the pureview was officially the last symbian phone, https://techcrunch.com/2013/01/24/nokia-confirms-the-pure-view-was-officially-the-last-symbian-phone/. January 2013. Accessed 27 Dec 2016
22. The Statista Portal. Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016 (2016), https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. Accessed 27 Dec 2016
23. I. Dalmasso, S.K. Datta, C. Bonnet, N. Nikaein, Survey, comparison and evaluation of cross platform mobile application development tools, in *9th International Wireless Communications and Mobile Computing Conference (IWCMC)* (IEEE, 2013), pp. 323–328
24. The Rohmobile Community Portal. Rohmobile suite documentation, http://docs.rhomobile.com/en/5.4/home. Accessed 28 Dec 2016
25. The Wikipedia Community Portal. Appcelerator titanium, https://en.wikipedia.org/wiki/Appcelerator_Titanium. Accessed 28 Dec 2016
26. The Wikipedia Community Portal. Apache cordova, https://en.wikipedia.org/wiki/Apache_Cordova. Accessed 28 Dec 2016
27. R. Droms. Dynamic host configuration protocol (1997), https://tools.ietf.org/html/rfc2131 (RFC 2131)
28. S. Deering (ed.), The author team of RFC-1256. ICMP router discovery messages, http://www.ietf.org/rfc/rfc1256.txt. September 1991 (RFC-1256)
29. C. Perkins, Minimal encapsulation within IP, http://tools.ietf.org/html/rfc2004. October 1996 (RFC-2004)
30. V. Jacobson. Compressing TCP/IP headers for low-speed serial links, https://tools.ietf.org/html/rfc1144. February 1990 (RFC-1144)
31. W. Simpson, IP in IP tunneling (1995), https://tools.ietf.org/html/rfc1853 (RFC 1853)
32. C. Perkins, IP encapsulation within IP (1996), http://tools.ietf.org/html/rfc2003 (RFC-2003)
33. G.E. Montenegro, Reverse tunneling for mobile IP, revised (2001), https://tools.ietf.org/html/rfc3024 (RFC 3024)
34. G. Bhaskara, A. Helmy, TCP over micro mobility protocols: a systematic ripple effect analysis, in *IEEE 60th Vehicular Technology Conference, VTC2004-Fall*, vol. 5 (IEEE, 2004), pp. 3095–3099
35. T. Krovetz, P. Rogaway,The software performance of authenticated-encryption modes, in *The 18th International Conference on Fast Software Encryption, 2011 (FSE 2011)*, pp. 306–327
36. Ted Krovetz and Phillip Rogaway. The OCB authenticated-encryption algorithm (2014), https://tools.ietf.org/html/rfc7253
37. O.M. Alliance, Technical_wap2_0_20021106 (2002), http://technical.openmobilealliance.org/Technical/technical-information/material-from-affiliates/wap-forum#previous
38. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext transfer protocol-HTTP/1.1, https://www.ietf.org/rfc/rfc2616.txt. June 1999 (RFC-2616)
39. Tim Berners-Lee and Team of Contributors of HTML. Html living standard (2016), https://whatwg.org/pdf
40. Wireless Application Protocol Forum. Wireless application protocol wireless markup language specification (2000), http://technical.openmobilealliance.org/tech/affiliates/wap/wap-238-wml-20010911-a.pdf (Version 1.3)
41. World Wide Web Consortium et al., Html 4.01 specification (1999)
42. A. Rauschmayer, in *Speaking JavaScript: An In-depth Guide for Progammers* (O'Reilly, 2014)
43. D. Singelée, B. Preneel, The wireless application protocol (WAP). Cosic Internet Report (2003)