Multimedia Tools and Applications, 25, 217–251, 2005

Bringing the Wireless Internet to UMTS Devices: A Case Study with Music Distribution

MARCO ROCCETTI roccetti@cs.unibo.it
PAOLA SALOMONI salomoni@cs.unibo.it
VITTORIO GHINI ghini@cs.unibo.it
STEFANO FERRETTI sferrett@cs.unibo.it
Department of Computer Science, University of Bologna, Mura A. Zamboni 7, 40127 Bologna, Italy

Abstract. Wireless networking is becoming an increasingly important communication means. Entry points to content delivery networks (e.g. portals) have to cope with the primary necessity of distributing multimedia contents to 3G mobile devices. In this paper we discuss the software architecture of a wireless Internet application we have designed and implemented to support the distribution of Mp3-based songs to 3G UMTS devices. Alongside the operational description of the proposed architecture, efforts have been made to examine the effects that Internet traffic has on the performance of UMTS networks, due to the distribution of Mp3 files by means of our wireless application. The download time measurements we have experimentally obtained show that combining modern 3G mobile network technologies with an appropriate structuring of the wireless Internet application may be very effective for the *fast* distribution of pre-recorded music to mobile clients.

Keywords: digital media, music on demand, wireless multimedia applications, CDN, UMTS

1. Introduction

Wireless networking is becoming an increasingly important communication means, even if wide-area wireless data connectivity is still difficult to achieve due to various technological limitations. The possibility that many Internet users will soon access e-mail and Web pages through a PDA or a smart phone is increasing very quickly [29]. Moreover, besides *traditional* electronic services, other important categories of multimedia services are emerging, including real time entertainment (e.g., movies and songs), interactive games (e.g., lottery, karaoke and distributed networked games) and infotainment (e.g., distributed learning and interactive consultations of multimedia information), whose multimedia contents must be delivered to both wired and wireless clients.

Simply stated, entry points to multimedia content networks have to cope with the primary necessity of enabling multimedia-based content distribution to mobile terminals. Since it is expected that there will be soon a greater demand for a mobile access to those advanced Internet-based multimedia wireless services, it is easy to envisage that the success of the new communication technologies will depend on how efficient the wireless access to those Internet-based multimedia services will be.

It is well known that the most widely used standard for second-generation (2G) mobile radio networks is the *Global System for Mobile Communications* (GSM). With the

introduction of the *General Packet Radio Service* (GPRS) for mobile networks operating under GSM, packet data connections have been allowed with variable bit rates up to several tens of Kb/s. Third-generation (3G) mobile systems, using the *Universal Mobile Telecommunications System* (UMTS), will offer higher data rates (of up to a few Mb/s) and an increased capacity. These data rates plus compression techniques will allow the access to HTML pages, to video/audio streaming, as well as to enhanced multimedia services for laptops and smaller devices.

Specifically, the UMTS forum describes four different traffic classes of possible services, whose quality is as follows:

- *Conversational class*, for supporting traditional real-time services, like real time telephony and video-telephony, with stringent and low delays.
- *Streaming class*, for supporting real-time traffic flows that need to preserve time relation between different entities of the stream, e.g. video on demand.
- *Interactive class*, for providing support to interactive best effort traffic such as traditional Internet applications (e.g., Web browsing),
- *Background class*, for supporting non-interactive best-effort traffic, such as, e.g., simple download of electronic mails or files.

An important advantage of these new mobile network technologies (including both GPRS and UMTS) is that packets originating from GPRS/UMTS mobile devices can be directly transmitted to data networks based on the Internet Protocol (IP), and vice-versa. This is due to the fact that GPRS/UMTS networks support special *border nodes* (termed GSN) that use IP as the backbone protocol for transfer and routing of protocol data units [46].

However, in this communication scenario several important problems arise. The major problem is to decide if advanced TCP/IP based applications will be able to behave well over wireless radio links [26, 44]. TCP was especially designed for wire-based transport and the protocol design rests upon on a number of assumptions that are typical of the wired environment. The heart of these assumptions is that unexpected increases in delay are interpreted as packet losses caused by network congestion. To respond to perceived losses, standard TCP aggressively slows its transmission to allow the network to recover [13, 43]. Unfortunately, wireline and wireless networks are significantly different in terms of bandwidth, propagation delay and channel robustness. In a wireless environment, in fact, TCP transmissions are typically prone to types of delays and losses that are not directly related to congestion, but to motion. If motion is mistaken for congestion and, consequently, normal congestion control procedures are triggered, then further performance degradation may be experienced during transport-level connections [21]. To alleviate this problem, an host of techniques, e.g., [1–5, 11, 12, 17, 20, 35, 42, 48, 49], has been proposed in the literature that addresses specifically the issue of making TCP suitable for the mobile wireless environment.

However, another important problem here is that many of these proposals either entail relevant changes in the TCP stacks within the Internet and in the mobile devices, or do not permit the mobile devices a transparent access to the Internet.

After having considered all the above challenges, a final problem is related to the internal architecture of those advanced Internet-based applications that should be accessed through radio interfaces. Those applications, in fact, must exhibit a high rate of robustness and availability, since the mobile access to those applications should not be influenced by possible problems occurring at the Internet side.

In this context, we have designed, developed and experimentally evaluated a wireless Internet software application that implements a *mobile music-on-demand service* to be enjoyed on UMTS devices. The developed application permits to mobile users to download and to listen to Mp3 files [32] through UMTS devices. Specifically, our wireless application exploits the *background* traffic class of UMTS to provide its users with: (i) a simple and rapid Internet-based mobile access to a music-on-demand download service, and (ii) a robust and widely available music-on-demand distribution system based on the technique of replicated Web servers.

From a user's standpoint, it is worth noticing that different types of clients may exploit our developed system. In particular our wireless application may be exploited by:

- *Music listeners*, they are single clients, equipped with a mobile UMTS device and connected to their UMTS cell, who may want to search for their favorite songs over the Internet, download them onto their UMTS devices, and finally play them out at their earliest convenience.
- *Music producers,* they are single clients, who may wish to exploit the system in order to distribute their own music songs to be listened to on UMTS devices. At the current state of the art of our system, this kind of users needs a regular wireline Internet connection in order to upload to the system their Mp3 music resources.
- *Musical service providers*, they may exploit the system to organize, build and maintain structured repositories of Mp3 resources over the Internet for use from UMTS devices.

The important experience of systems for the distribution of contents over the Internet (i.e., Content Distribution Networks) have inspired our work [3, 6, 8, 9, 14, 16, 19, 33], but our system is essentially new, in the sense that it allows a reliable and distributed song delivery service for mobile UMTS devices. In particular, in order to ensure both the availability and the responsiveness of our music-on-demand service, we have structured our system according to the special technology of *replicated Web servers* [7, 22]. In essence, according to this technology, a software redundancy is introduced at the Internet side, namely by replicating the music songs composing the music-on-demand service across a certain number of Web servers which are geographically distributed over the Internet. In this context, a typical approach to guarantee service responsiveness consists of dynamically binding the service client to the available server replica with the least congested connection.

An approach recently proposed to implement such an adaptive downloading strategy at the Internet side amounts to the use of a software mechanism, called the Client-Centered Load Distribution (C^2LD) mechanism [15]. With this particular mechanism, we split each client's request of a given musical resource (i.e., a song) into a number of sub-requests for separate parts of the resource. Each of these sub-requests is issued concurrently to a different available replica server, which possesses that song. The mechanism periodically

monitors the downloading performance of available replica servers and dynamically selects, at run-time, those replicas to which the client sub-requests can be sent, based on both the network congestion state and the replica servers' workload.

As far as the protocol communication problems mentioned above are concerned, our wireless application has been structured based on the use of an *ALL-IP* (or open IP) approach where the mobile UMTS device is allowed to function as any other Internet-connected device [21], and an end-to-end direct TCP/IP continuity is guaranteed by exploiting a standard TCP/IP protocol stack. The principal motivation behind our choice is that of providing seamless internetworking between the wired and the wireless segments.

Additionally, as one of the most relevant problems for music distribution to mobile devices is that of an unexpected link interruption in the midst of a long download activity, we have equipped our wireless architecture with a session level developed on the top of the standard TCP protocol. The goal of our session level is that of ensuring that the song download activity is not interrupted when the TCP communication is destroyed due to very long link outages or handoffs. As explained in the following Section 2.2, here the problem is no longer one of adjusting TCP performance to match the requirements of the wireless environment, but one of guaranteeing a successful termination of the download activity even when the underlying TCP connection is destroyed due to device mobility. Based on the consideration that our session mechanism has been designed only to recover from stable damages experienced by TCP connections, it is possible to imagine a future extension of our system that includes one of the TCP wireless protocols which were proposed to alleviate the worst aspects of performance degradation of the standard TCP protocol stack [e.g., 1, 5, 17].

Our paper concentrates on two objectives. First, it provides a complete overview of the wireless Internet application we have designed and implemented; second, it presents a complete set of experimental results that exhibit the performance of our system. The reminder of this paper is organized as follows. In Section 2, we provide a detailed description of the architecture of the wireless Internet application we have developed, and discuss some design choices we have taken to implement our system. Section 3 reports on a large set of performance results we have gathered from real-world experiments. In Section 4, some related work by other researchers is presented and compared with the design goals at the basis of our system. Finally, Section 5 gives some concluding remarks and plans for future work.

2. System architecture

The general architecture of our proposed wireless application may be thought of as constructed out of the following three software components, as shown in figure 1:

- The *Mobile Client Application*. This part of our wireless application runs on the UMTS device, and has the responsibility of supporting the client during the subsequent activities of:

  1. Searching the Mp3 files corresponding to the client's favorite songs over the Internet,
  2. Downloading them on the UMTS device,
  3. Playing them out as soon as they have been downloaded.

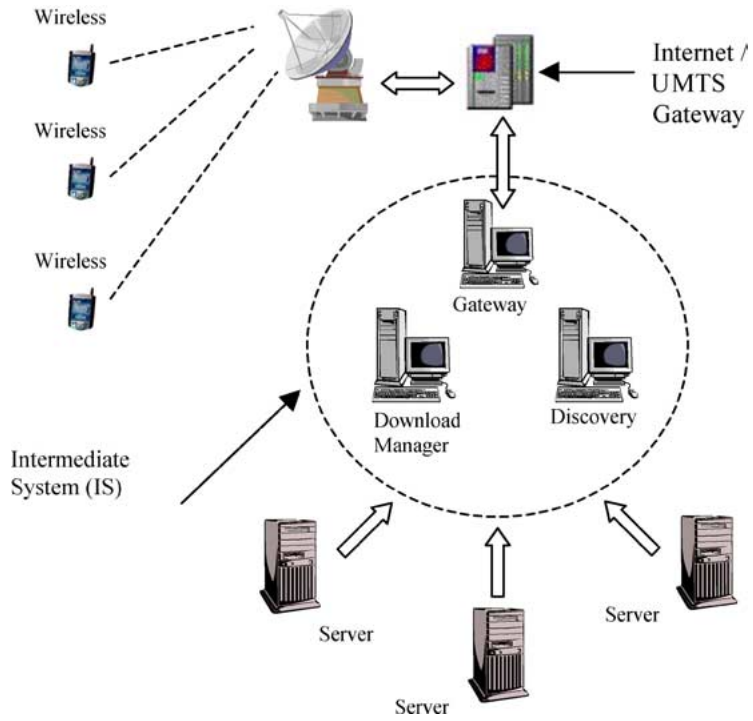

Figure 1. System architecture.

- The *Intermediate software system* (IS). This software component is hosted in an Internet server and represents the core of our system. It is in charge of managing all the communications between the UMTS device and the Internet infrastructures. In particular, the IS has been designed to support:

  1. The *wireless communication* with the UMTS device, providing it with the access point to the Internet-based music distribution service;
  2. The *wireline* communication with the replicated Web servers. In essence, it is in charge of implementing a reliable and responsive discover-and-download service based on the user's requests.

- The replicated *Web Servers*. They are Web servers geographically distributed over the Internet which act as music repositories. In general, different Web servers can be managed and administrated by different music service providers, and may also offer different set of replicated songs. Simply stated, this replication scenario can be thought of as a *loosely coupled* replication system, where, potentially, different servers support different sets of music resources, and each single musical resource may be replicated within a number of geographically dispersed Web servers.

As to the IS, it exhibits three main functions which correspond, in turn, to the three following software sub-components of the IS:

- the *Application gateway*; the Application Gateway accepts and manages all the requests for songs arriving from the client connected to a given UMTS terminal,
- the *Discovery system*; it has been developed to identify and locate within our Internet-based system all the replicated resources (i.e., the replicated Mp3 files) which correspond to songs that have been requested by the users,
- the *Download manager*; it has been developed to carry out the activity of downloading from the Web servers to the IS the songs that have been identified by the Discovery system. The Download Manager performs this activity by exploiting the C^2LD download mechanism mentioned earlier.

A complete search-and-download session works as follows. Initially, a user from his/her UMTS device issues to the Application Gateway a request for a given song. (Such a request may refer either to a given song title or to a specific author). The Application Gateway passes this request down to the Download Manager. The Download Manager asks to the Discovery for the complete list of all the available MP3 songs which match the request issued by the user. The Discovery subsystem performs the research of the song required by the client.

Such an activity steps trough two different phases and proceeds as follows. First, the Discovery tries to establish a relationship between the title of the song requested by the user and the set of the available MP3 songs which can match the request. (Note that different MP3 songs may exist that match the request issued by the user). On completion of this operation, the Discovery passes to the user (via the Application Gateway) the list of all the MP3 songs it has found. Upon receiving this list, the user chooses one of the proposed MP3 songs. This choice triggers an automatic process to download the correspondent song. The Download Manager asks to the Discovery the complete list of the Internet-based locations of all the replicated files that correspond to the MP3 song chosen by the user. After that, the Download Manager starts the download operations by engaging all the different replica servers that maintain identical copies of the requested song. This represents the beginning of the second phase of the discovery/download activity, at the end of which the reassembled Mp3 file is sent back from the Download Manager to the Gateway, and from the Gateway to the UMTS terminal.

Needless to say, in order for the system to work correctly, a preliminary phase has to be carried out where each potential music server announces the list of musical resources it wishes to make available for distribution. Each music server which wants to add its own repository to our IS may do that by running a software application called the *Data Collector*. This application is in charge of communicating to the Discovery the list of the Mp3 songs currently offered by different music servers. The next Subsections are devoted to examine, in turn, each of the above mentioned software application components, along with a number of relevant design and implementation details.

2.1. The Mobile Client application

The Mobile Client application represents the interface between human users and system services, and provides a set of *search-and-download* functions. Taking into account that portable devices (such as UMTS telephones or PDAs) are both computing-and communication-power limited, we have taken the decision to delegate all the search and discovery functions to the IS. In other words, the Mobile Client application cannot autonomously determine which Mp3 files are precisely available, and where they are located, but has to perform the two-phased search/download activity we mentioned earlier.

In the first phase, the mobile user initiates a search for a song, providing the system with the request for a given song or an author. Hence, the Mobile Client application contacts the IS to verify the existence of the corresponding music resource within the system. If that resource exists, then this information is given back to the user. In the second phase, the user can start the download activity to obtain the musical resource that has been previously identified.

At the current state of the art, our mobile client application is delivered over the Microsoft Windows CE operating system [31]. The only rationale behind this choice is that the Windows CE platform provides a plethora of different programming tools, such as Visual C++ and Visual Basic, for example. Future implementation efforts will be devoted to develop our client application over different operating systems (such as Linux). The current version of the client software application has been implemented in Visual C++ by exploiting the network programming environment provided by the socket programming interface.

It is worth noticing that the client software application we have developed permits to the user the access to the music download service via a graphical interface through which she/he can: (i) issue requests for a given song, (ii) ask for the top-10 list of songs, (iii) download songs, (iv) listen to the downloaded songs. An example illustrating the final download phase of a complete *search/download/playout* process is provided in figure 2.

2.2. The Application Gateway

The Application Gateway is the IS component that receives requests from the mobile terminal and redirects them to the Download Manager. Finally, it forwards the downloaded file to the mobile client over a wireless link. It is under the responsibility of the Application Gateway that our system communicates with the mobile device.

Figure 3 shows the protocol stack we have developed to support all the Application Gateway-related communications. In particular (as shown in the leftmost side of the figure) the Application Gateway communicates with the Mobile Client application over an UMTS link. As seen from the figure, on the UMTS protocol stack an IP layer is implemented, based on the MobileIP (version 4) protocol. On the top of this MobileIP level, a standard TCP layer has been built. Finally, the application layer built on the top of TCP has been designed as constructed out of two different sub-layers:

- a *Session Layer*; this protocol layer is devoted to manage a download session which may provides users with the possibility of resuming a communication that was previously

Figure 2. A download playout session.

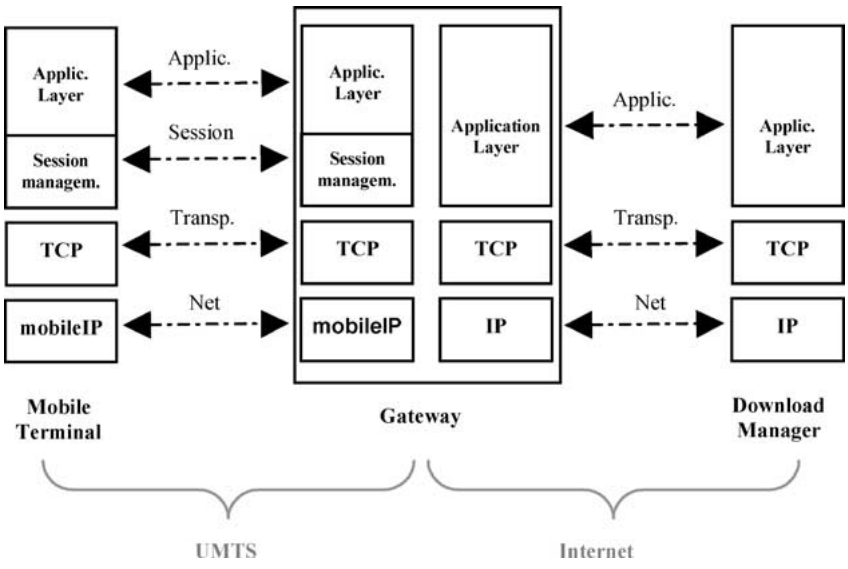

Figure 3. The application gateway protocol stack.

interrupted due to problems occurring at the lower levels of our communication architecture,

- a *(Real) Application Layer*; this protocol layer is in charge of supporting the different connections needed to search and to download songs.

It is well known that mobile wireless is one of the more complex environments for traditional transport protocols, such as TCP. Standard TCP has been designed to interpret unexpected delays and packet losses as signs of network congestion. The normal TCP's response to packet losses and delays is to trigger a back off procedure (called *slow start*) that aggressively reduces the congestion window to achieve connection stability [13, 43].

In a wireless environment, instead, unexpected delays and packet losses are symptoms of problems related to mobility, such as temporary link outages, handoffs and transmission errors due to bit-level corruption [21]. The heart of the problem is that, if mobility is mistaken for congestion and the aggressive TCP congestion control procedures are activated, the overall throughput of the wireless connection is drastically reduced with a significant performance degradation of the entire system.

To alleviate the performance problems experienced by the existing TCP stack over wireless links, a number of different solutions have been proposed that modify the standard TCP protocol [1, 2, 4, 5, 10, 12, 17, 20, 35, 42, 48, 49]. Unfortunately, drawbacks of practical relevance of many of those proposals are that either their implementation requires changes in the TCP stack within the Internet and in the mobile devices, or they do not permit the mobile devices a transparent access to the Internet.

In this scenario, to guarantee that our mobile wireless devices may function as any other Internet-connected device, we have taken the decision to resort to a so called *ALL-IP* (or open IP) approach that exploits a standard TCP-IP stack for the communications between the Application Gateway and the Mobile Client application. The principal motivation behind this choice is to ensure a full end-to-end TCP/IP continuity.

However, as one of the most relevant problems for music distribution to mobile devices is that of an unexpected link interruption in the midst of a long song download activity, we have equipped our architecture with the wireless *session layer* shown in figure 3.

The aim of our session level is no longer one of adjusting TCP performance to match the requirements of the wireless environment, but one of ensuring a successful termination of the download activity even when the underlying TCP connections are damaged or disrupted due to the device mobility.

A subtle aspect of TCP-supported wireless communications, in fact, is represented by the problems that may be caused by long link outages and handoffs. For example, a mobile user may enter an area of no signal coverage for a given period of time. Typically, if this period of time exceeds a given threshold (e.g., a few minutes) the network gateway located in between the wired and the wireless segments assumes that a link level interruption has occurred and, after having informed the mobile device, it destroys the data link connection over the wireless link. We have two possible cases here:

In the former case, the mobile client may have left the area of no signal coverage in time to receive the information that the communication has been interrupted at the link level. In such a case, the TCP software running on the mobile device may only take the decision

to destroy the TCP connection as no more data will be delivered to the mobile device through that TCP connection.

In the latter case, instead, the mobile client may still be in the area of no signal coverage when the network gateway tries to inform it that the link level communication has been lost. Under this circumstance, there is no way that the TCP software can be informed of the link level interruption and a further negative result is that the TCP connection remains open forever at the client side, even though no more data may be received over that TCP connection.

Both the cases mentioned above may be very critical as they may cause the interruption of the download activity of large Mp3 files, with no possibility to resume the data stream. Based on these considerations, it easy to recognize that only the presence of a session mechanism which is able to manage the communication interruptions we have illustrated may guarantee the full success of a song download activity. With this in view, we have designed a session management mechanism which is able to manage possible interruptions at the lower levels of the communication architecture even in the presence of a wireless cellular access which may exhibit a scarce connection stability and an unpredictable availability.

Our *session layer* works at the Client side as illustrated in figure 4, and summarized below:

When the UMTS Client Application opens a download session with the Application Gateway through a TCP connection, the Application Gateway assigns a unique identifier to this new session and transmits it to the Client (lines 1–2, figure 4). During the download activity (line 3, global variable: *download_completed*), if the TCP software running on

```
1 request_new_session()
2 sessionID:=receiveID()
3 while not(download_completed) do
4     if not(TCP_error or connection_timeout) then    /*read data from the current TCP
                connection unless a TCP error is intercepted or a timeout occurs*/
5           read()
6     else                          /*the link level connection is interrupted*/
7           save_session_state() /*save session state: session_ID and last_byte_received*/
8           suspend_session()       /*destroy old TCP connection at the Client side*/
9           while not(session_resumed) do /*try to resume session*/
10                sleep()                /*waiting for the link to become operational*/
11                resume_session() /*open a new TCP connection using the session_ID and
                                          the pointer to the last_byte_received*/
12          od
13    fi
14 od
15 close_session()                 /*download completed at the client, termination of
                                     session is communicated to the Application Gateway*/
```

Figure 4. Implementation of the session layer (Mobile Client).

the Client is informed of an interruption occurred at the wireless link level (line 4, global variable: *TCP_error*), or if a too long period of time has passed since the instant when the last byte was received by the Client (line 4, global variable: *connection_timeout*), the (download) session state is saved at the Client side. In particular, a pointer to the last byte received of the Mp3 song is saved along with the session identifier (lines 6–7), while the current TCP connection is destroyed since it has been presumably damaged due to a link level interruption (line 8).

As soon as the wireless link has become operational and the Mobile Client is able to open a new TCP connection with the Application Gateway, then the Client tries to resume the interrupted session by transmitting to the Application Gateway the session state that was previously saved (lines 9–11, global variable: *session_resumed*).

Eventually, when the Client detects that the download is successfully completed, the session is terminated (line 15). At this point, the termination of the session is communicated to the Application Gateway.

The session management mechanism implemented at the Application Gateway side works to keep the operations performed by the Application Gateway synchronized with the Client. In particular, as shown in figure 5, when the Mobile Client contacts the Application Gateway to download a song, the Gateway detects if this is either a request for a new download session or an attempt to resume an already existing session. (We have already mentioned that in this latter case the Client transmits to the Gateway both the session identifier and the pointer to the last byte it has received in that session).

If the Gateway is requested to open a new session (line 1 in figure 5, global variable: *new_session_request*), it assigns an identifier to this new session and, using the current TCP connection, transmits the session identifier to the Mobile Client along with the first data of the requested song (lines 2–3 and 8).

Instead, upon receiving a request to resume a session the Gateway first checks if an old TCP connection exists which is still open and inactive under the session to be resumed. In such a case, the Gateway destroys that old TCP connection as it has not any active counterpart at the Client side. Then, the Gateway resumes the session and starts sending data to the

```
1 if (new_session_request) then
2      open_session()
3      sendID(sessionID)              /*send ID identifier to the Client, ready to transmit
                                        data from the first byte*/
4 else
5      resume_session()               /*terminate old TCP connection, resume session ready to
                                        transmit data from last_byte_received*/
6 fi
7 while not(download_completed or session_timeout) do      /*write data onto the current TCP
                              connection unless the download is completed or a timeout occurs*/
8            write()
9 od
10 close_session()  /*session closed*/
```

Figure 5. Implementation of the session layer (Application Gateway).

Client using the most recently generated TCP connection. This upload activity commences with the byte that follows the last byte that was successfully received by the Client before the link disruption (lines 5 and 8). If, eventually, the Client informs the Application Gateway that the download activity has been successfully completed then the current session may be terminated at the Gateway side (lines 7 and 10, global variable: *download_completed*).

It is also worth noticing that the session layer implemented at the Gateway is equipped with an internal mechanism based on a timeout which may be used to autonomously terminate a session. This decision is taken by the Application Gateway when a considerable amount of time has passed without any communication coming from the Client about a given session (lines 7 and 10, global variable: *session_timeout*). In our prototype implementation the value of this timeout is set equal to one hour.

As a final note, it is important to remind that the session management mechanism we have developed is suitable for recovering download activities that are interrupted due to long link outages and handoffs, but it is not adequate to recover from system failures occurring at the UMTS terminal or at the Application Gateway.

2.3. The Download Manager

The Download Manager is the real agent responsible for the download process and has been designed to be able to optimize data transmissions in the sense that:

- It maximizes the service availability, i.e., it tries to maximize the percentage of successfully served requests.
- It maximizes the service responsiveness, i.e., it tries to minimize the time after which a requested song is successfully downloaded on the UMTS terminal.

The aforementioned objectives have been fulfilled, at the Internet side, by exploiting the technique of *replicated Web servers*. According to this technology, a software redundancy is introduced at the Internet side. In essence, the songs which compose a music-on-demand service are replicated across a number of Web servers, geographically distributed over the Internet. In this context, a typical strategy to guarantee service responsiveness and availability consists of dynamically binding the client to the available server replica with the least congested connection [22].

An approach recently proposed to implement such a download strategy at the Internet side amounts to the use of a software mechanism, called the Client-Centered Load Distribution (C^2LD) mechanism [15]. The principal goal of this mechanism is to minimize what we term the User Response Time (URT), i.e., the time elapsed between the generation of a request for the retrieval of a given Web resource and the delivery of that resource to the final user over the wireless link.

Simply stated, rather than binding a client to its "most convenient" replica server as proposed in [7], C^2LD intercepts each request for a music resource, and fragments that request into a number of sub-requests for separate parts (or fragments) of that resource. Each sub-request is issued to a different available replica server, concurrently. For each sub-request, an internal timeout is set, and if this timeout expires before the requested

fragment is received, the fragment is required from another replica server. Finally, the replies received from the replica servers are reassembled to reconstruct the requested song which is then delivered to the final user.

The C^2LD mechanism is designed so as to adapt dynamically to state changes in both the network (e.g., route congestion, link failures) and the replica servers (e.g., replica overload, unavailability). To this end, the C^2LD mechanism monitors periodically the available replica servers and selects, at run-time, those replicas to which the sub-requests can be sent, i.e., those replicas that can provide the requested song fragments within a time interval that allows the mechanism to minimize the URT.

We have implemented the C^2LD mechanism on top of the HTTP 1.1 interface. The timeliness requirement that our C^2LD mechanism has to meet, in order to be effectively responsive, can be expressed by means of a Intermediate System Specified Deadline (IS^2D), i.e., a value that indicates the extent of time our Intermediate System (IS) is willing to wait for a requested music resource to be fetched from the Web.

For the purposes of this implementation, we have provided our mechanism with a configuration procedure that allows the IS to set the IS^2D timeout before a music resource is accessed (a default IS^2D value is used by our C^2LD implementation, if the configuration procedure is not exploited).

Typically, in order to obtain a song, a mobile user starts a request by providing the name of the song. When the request for a given song is received at the IS, a translation of this request is performed by the Discovery System that returns the IP addresses of all the Web replica servers that store that song. At that precise instant, the Download Manager may use standard HTTP GET methods with the IP addresses of all the replica servers associated to that song. The C^2LD mechanism intercepts each HTTP GET invocation, starts the IS^2D timeout, and uses the IP addresses in the GET invocation to interact with each replica as illustrated in figure 6, and summarized below.

In particular, C^2LD invokes an HTTP HEAD method on each replica i. The reply from replica i to the first HTTP HEAD invocation is used by C^2LD to: (i) get the exact size of the requested song, (ii) estimate the data rate (DR_i) that replica i can provide, and (iii) assess the fragment size (FS_i) of the first fragment that can be fetched from that replica (lines 3–5 in figure 6).

C^2LD maintains a global variable (*download_done*, line 8 in figure 6) that indicates whether or not all the fragments of a requested song have been delivered. Before proceeding in our discussion, it is worth observing that, owing to the IS^2D timing constraint, each replica server that receives a sub-request for a song fragment must honor that sub-request within a time interval that allow the C^2LD mechanism to reconstruct the requested song, out of all the received fragments, before the IS^2D deadline expires. Thus, it is crucial that the C^2LD mechanism assesses accurately both the size of the fragment each replica is to supply, and the time intervals within which these fragments are to be received at the IS site.

To this aim, until a song is not fully downloaded, C^2LD periodically computes the size of the fragment ($FS_{i,r}$) to be requested to the replica i within a period r of expected duration equal to $D_{i,r}$, using the following formula:

$$FS_{i,r} = DR_{i,r} * D_{i,r.}. \tag{1}$$

```
   …                /*intercept song request*/
   …                /*interrogate Discovery for replicas IP addresses*/
1 within IS²D do                /*set IS²D timeout*/
2    for each_replica do
3      HEAD();                  /*send HEAD request to replica i to get song size*/
4      DRi,1:= …;   /*use response time to HEAD to compute the data rate DRi,1 for replica i*/
5      FSi,1:= …;   /*compute first fragment size for replica i*/
6      TIME_OUT:= …;                    /*compute time out for fragment fetching*/
7      within TIME_OUT do               /*set timeout of length TIME_OUT*/
8      if not download_done then        / check if song downloaded*/
9             GET();                   /*get fragment from replica i*/
10            DRi,r:=FSi,r-1 / DTi,r-1 ;  /*compute expected Data Rate*/
11            FSi,r:=DT i,r . Di,r;       /*compute fragment size of next fragment*/
12     else
13            return;                  /*song downloaded at the IS*/
14     fi
15     od
16   od
17 od
```

Figure 6. Implementation of the C²LD Service.

In the Formula (1) above $\mathrm{DR}_{i,r}$ represents the value of the most recently measured data rate that can be estimated by means of the following equation:

$$\mathrm{DR}_{i,r} = \frac{\mathrm{FS}_{i,r-1}}{\mathrm{DT}_{i,r-1}}. \tag{2}$$

In the above equation, $\mathrm{FS}_{i,r-1}$ represents the size of the song fragment downloaded with the previous sub-request $r-1$ issued to replica i, while $\mathrm{DT}_{i,r-1}$ is the download time experienced for downloading the $r-1$ song fragment of size $\mathrm{FS}_{i,r-1}$. (Note that the $\mathrm{DT}_{i,r-1}$ value may be experimentally measured at the time of completion of the previous sub-request $r-1$). Once the requested fragment size has been calculated, C²LD issues a HTTP GET request to the replica i to retrieve the required fragment (lines 9–11 in figure 6).

It is worth noticing, here, that no modification is needed to the standard storage of sequential MP3 files on the Web servers, due to fragmentation. Specifically, a fragment of Z bytes size may be requested by invoking a standard HTTP GET method with the following option set: "*Range: bytes* $= Y$–X", where X and Y denote the bytes corresponding to the beginning and the end of the requested fragment of size Z, respectively.

In essence, to adapt to possible fluctuations of the communication delays that may occur over the Internet, each time a fragment is to be requested, its size is computed based on the value of the download time $\mathrm{DT}_{i,r-1}$ experienced in fetching the previous fragment. Thus, as the GET request $r-1$ directed to a given replica i terminates, a new GET request r can be issued to the replica i with the fragment size value $\mathrm{FS}_{i,r}$ computed on the basis of the previous download time.

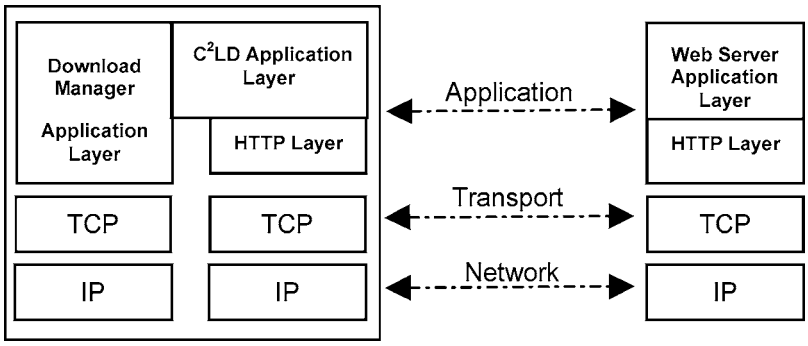

Figure 7. The download manager protocol stack.

Note that some of the replica servers may not respond timely to the HTTP (HEAD and GET) invocations described above (e.g., they may be unavailable owing to network congestion). Thus, C^{2}LD associates a timeout to each HTTP request it issues to each server i (line 6 in figure 6). If that timeout expires before C^{2}LD receive a reply from a replica server i, it assumes that the server i is currently unavailable, and places it in a "stand_by" list. Replica servers in that list are periodically probed to assess whether they have become active again. Requests for replicas in the *stand_by* list are redirected to other active replicas [7].

We conclude this Subsection by reporting in figure 7 the Download Manager protocol stack. As the Download Manager has to communicate simultaneously with different mobile clients, it has been designed to fork into different children processes for each different request. Each process then uses the C^{2}LD mechanism to download Mp3 files from the Web server replicas, as shown in the figure.

It is also worth mentioning that the use of the C^{2}LD mechanism does not force music providers to organize Mp3 repositories which are all perfect replicas of the same list of songs. A song, in fact, may be replicated within only some of the available server replica of our system. The way in which our system is able to determine if a requested song is in existence in the system, and where it is replicated, is discussed in the following Subsection.

2.4. The discovery and the data collector

The software component of our system where the relevant information about songs are stored and indexed is the *Discovery* system. The main responsibility of the Discovery is that of performing a sort of *naming resolution* for musical songs which are requested by clients. In particular, it is in charge of:

- establishing a formal relationship between the requested songs and the correspondent Mp3 files stored in the system,
- identifying the exact Internet locations where Mp3 files are replicated.

It is easy to understand that song titles and correspondent authors are useful for file indexing, but are not sufficient for accurate file identification. For example, the same song may be encoded at different sampling rates thus resulting in different Mp3 files, or identical Mp3 files may exist with different names or titles. To overcome this kind of problems, the Discovery is able to identify identical Mp3 copies of a given song by calculating a 32 bit-based identifier (called the *checksum)* which is computed on the basis of the file content. In essence, the complete list of information managed by the Discovery are the following:

- The information needed for identifying the set of Mp3 files corresponding to a given song, namely: the *File Name*, the *File creation time*, the *File length*, the *Song Title* and *Author*, and finally the *Checksum*.
- The information needed for localizing different identical Mp3 copies of the same song, namely: the *File Name*, the *Host server address* (the IP address of the server that maintains the requested song), the *Path* (along which the file is stored in the host server), the *Transfer Application Protocol* used for distribution (e.g., HTTP) and finally the *Checksum.*

Specifically, the Discovery system manages two different hash indexes: the former, needed to resolve user's requests, is created on the basis of the song title and author while, the latter, used to localize the requested files, is created on the basis of the checksum value mentioned above. Two alternative methods may be devised for performing the calculation of the checksum:

- a *Centralized* method, according to which each MP3 file is transmitted from the host server to the Discovery that, in turn, computes the checksum, and
- a *Distributed* method, where each host server computes the checksum of its musical files and communicates the results to the Discovery system.

To minimize the traffic overhead, we have taken the decision to implement the distributed method, where each server has to locally run a software application, termed the *Data Collector*, which provides the possibility to add or to delete the songs to be referenced by the Discovery system. In this case, it is the responsibility of the Data Collector to locally perform the checksum computation. The Data Collector is implemented as a Java applet to enhance the software portability, and also meets standard security constraints, as it can only read from the local file system, but it cannot execute local write operations. After having computed the checksum of all the files that a given music provider wishes to distribute, the Data Collector opens a TCP connection towards the Discovery and uploads the computed checksums to it.

It is worth noticing that our signature mechanism has been specifically designed to cope with the fragmentation/reassembly process provided by the C^2LD mechanism which needs to work with identical Mp3 copies of a given song. Based on our experience with other systems designed to distribute music over the Internet (e.g., Napster [33]), we feel that our signature mechanism and the fragmentation/reassembly process cannot be considered as a limitation, but an important contribution to accelerating the activity of music distribution.

Typical interactions with Napster, in fact, were as follows. Different versions of a given song (say song A) were often present within Napster depending, for example, on

the bit rate used for sampling (e.g., songA-version1.mp3, songA-version2.mp3, songA-version3.mp3, ...). Several identical copies of a given Mp3 song were also frequently offered by different owners (e.g., five identical copies of songA-version2.mp3, and three identical copies of songA-version3.mp3). Napster clients were used to choose a given version of a certain song based on their preference and convenience. After having chosen a given version of a song (e.g., songA-version2.mp3), the download of that Mp3 song was performed by establishing a sequential HTTP connection with only one out of the different servers that stored the song of interest.

With our signature mechanism, instead, it is possible to build a replicated repository for each different version of a given song. Each replicated repository can be built with the contribution of each different owner of an identical copy of a given Mp3 song. For example, a replicated repository for songA-version2.mp3 could be set up with the contribution of all the five different owners of that Mp3 song. The real advantage of our approach is that:

- the construction of such a replicated repository may be carried out following a distributed and independent process, where each owner may use the Data Collector and its signature mechanism to add/delete its copy of a given Mp3 song, and
- a rapid song distribution is ensured by the C^2LD mechanism which is able to exploit all the available identical copies of a given Mp3 song.

3. An experimental assessment

We present an experimental study we have developed in order to assess the effectiveness of our music-on-demand service. The intention behind our experimental study has been to investigate the quality of the Internet/UMTS download sessions carried out by our wireless application. During the period July 2001–May 2002, we conducted around 4000 experiments consisting in the download of a set of different Mp3 files. Four different Web server replicas were exploited at the Internet side. Instead, the communications between the Application Gateway (located at the border between the Internet and the wireless UMTS link) and the mobile client was simulated by means of an UMTS simulator able to produce the transmission delay time of each frame at the radio link layer. Detailed information concerning the experimental models we adopted for our experiments are discussed in the following Subsections.

3.1. Application level model

We used four different Web servers, geographically distributed over the Internet, providing the same set of 40 different songs. The four different replica servers were respectively located in Finland, Japan, USA and New Zealand (figure 8). The Intermediate System was running on a Pentium 3 machine (667 MHz, 254 MB RAM) equipped with the Windows 2000 Server operating system, and was located in Italy (Bologna). The UMTS network was simulated by means of an UMTS simulator provided by the "Fondazione Marconi" (a public Italian foundation for wireless computing). Finally, the UMTS device, on which the

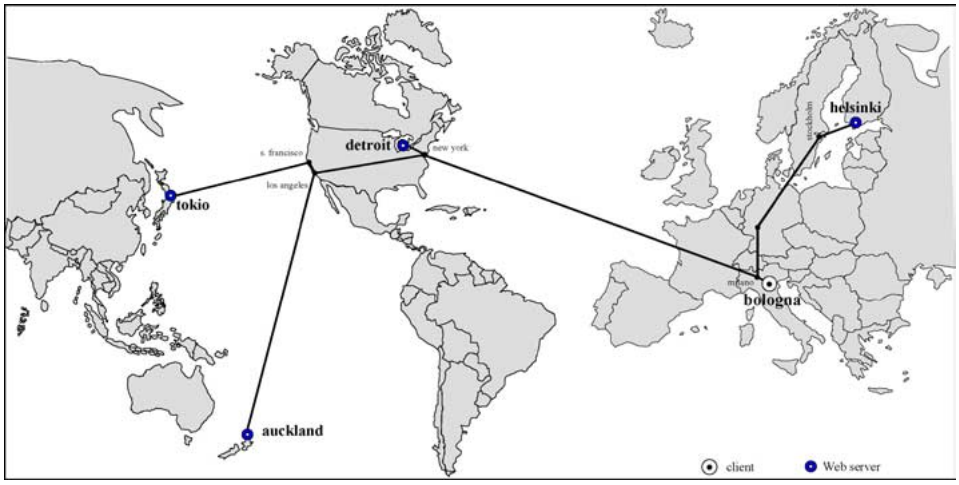

Figure 8. Web server replicas and clients.

client of our application was running, was emulated by means of a Pentium 2 computer (266 MHz, 128 MB RAM) equipped with the Windows CE operating system.

To provide the reader with an approximate knowledge of the transmission times experienced over the considered Internet links, it is worth mentioning that the average measurements, obtained with the ping routine, of the Round Trip Times between the client and the four different servers (i.e., Finland, Japan, USA and New Zealand) were 70, 393, 145 and 491 milliseconds, respectively. As to the downloading process, the two following basic assumptions were taken:

1. *Mp3 file size:* we used 40 different Mp3 songs whose correspondent file sizes ranged from 3 to 5 MB. The file size of 3-5 MB corresponds to the average file dimension of the songs maintained in the Napster system.
2. *Type of download activities*: our software application is able to support to two different types of download services: the former consists of downloading a single song, the latter amounts to the download of a complete set of songs (a compilation). To evaluate the performance of our system under both these circumstances, we conducted the following experiments:

   - A set of independently replicated experiments consisting of the download of a single song.
   - A set of independently replicated experiments, each one consisting of the download of a compilation. The number of songs for each compilation was chosen equal to either 3 or 5 or 10. These three values were chosen based on the consideration that the average disk capacity of typical Mp3 players never exceeds 50 MB.

3.2. TCP/IP and UMTS models

As currently no real measurements of UMTS data transmissions are available, the communication between the Application Gateway (at the Internet side) and the client application running on the UMTS device was carried out through a simulated UMTS network using the *background* traffic class.

It is well known that the UMTS protocol stack consists of: a PHY (Physical) layer, a MAC (Medium Access Control) layer, an RLC (Radio Link Control) layer that implements an ARQ mechanism for ensuring reliable data transmission, and finally, a PDCP (Packet Data Convergence Protocol) layer that provides data and header compression to improve channel efficiency [46]. On the top of this UMTS stack we have the standard IP and TCP protocols [13].

The UMTS network simulator we exploited is able to return after simulations a complete *Wireless Network Transmission Time* (WNTT) value computed at the PDCP layer. Needless to say, these WNTT values depend on some operational parameters, such as the amount of traffic present in the cell and the number of active clients and their speeds. WNTT measurements include also the time spent for possible retransmissions at the UMTS RLC level. In our experiments, different values of this WNTT measurements were taken based on the different possible sizes of the TCP segments coming from the Internet (namely 120/440/920 bytes).

The unique problem that stems from this hybrid approach (i.e., both experimental and simulative) is that segment errors and resulting retransmissions at the TCP level are not taken into account. To circumvent this problem, our experiments have included the possible retransmission time delays incurred at the TCP level, by exploiting an external delay introduction mechanism that was designed to take into account the typical TCP error recovery mechanism based on received ACKs. Simply stated, this delay mechanism compares the WNTT values obtained through the UMTS simulation against the *timeout* values computed by TCP. If the simulated WNTT value is larger than the correspondent TCP timeout value, then we conclude that a retransmission must occur at the TCP level. In such a case, the WNTT value of that given TCP segment is augmented by an additional value which is chosen as equal to the next WNTT value extracted from the set of the UMTS-based simulated values. Consequently, the TCP timeout value is updated as follows. If a retransmission at the TCP level is detected according to the method mentioned above, then the subsequent TCP timeout value is calculated as double with respect to the previously computed value. If no retransmission at the TCP level has been detected, then the traditional adaptive formula for the calculation of the TCP timeout value is followed [13, 43]:

$$
\begin{aligned}
&\mathit{Timeout} = \mathit{RTT} + 4 * D,\\
&\mathit{RTT} = \alpha * \mathit{RTT} + (1 - \alpha) * M,\\
&D = \alpha * D + (1 - \alpha) * |\mathit{RTT} - M|,
\end{aligned}
$$

where $\alpha = 7/8$, M is the simulated value produced by the UMTS simulator, *RTT* represents an averaged value of M, and D is a variation of *RTT*.

Table 1. WLNTT and SA results.

	C^2LD (4 Servers)	HTTP			
		Finland	USA	Japan	New Zealand
Download time (Seconds)	32.547	47.889	122.191	248.740	624.195
C^2LD improvement (Percentage)	–	32%	73.4%	86.9%	94.7%
Successful download (Percentage)	100%	98.5%	99.5%	95%	89%

3.3. Experimental results

This section reports on a large set of results obtained during the experimental trials we conducted. In particular, in the following section, we present both the download times and the service availability level we measured at the Internet side. Instead, Sections 3.3.2 and 3.3.3, respectively, present the measurements of the download times obtained for the distribution of single songs and for the distribution of sets of songs over an UMTS link.

3.3.1. Download times at the internet side. In this Section we report the measurements of the:

- *WireLine Network Transmission Time* (WLNTT) values, that is the time spent over the wired Internet links to download a requested Mp3 song from the Web server replicas to the IS. These measurements have been compared with those that may be obtained by downloading the same Mp3 song with a standard HTTP GET method. The first row of Table 1 reports those results for Mp3 files whose size is 5 MB. The second row shows the average WLNTT percentage improvement obtained by the C^2LD mechanism with respect to the standard HTTP protocol. As shown in the Table, our system obtains an average percentage improvement over the fastest HTTP replica which is equal to 32%;
- *Service Availability* (SA) values, i.e., the capability of carrying out a successful download of the requested song (within a maximum time interval of 900s). As shown from Table 1, a full SA may be achieved with the use of the replication technology adopted by our C^2LD mechanism. On the contrary, only a partial SA may be obtained by exploiting the standard HTTP protocol.

3.3.2. Download times for single songs on UMTS links. We examine here the cumulative *Wireless Network Transmission Time* (WNTT) values we obtained to download single Mp3 songs to UMTS devices. In particular, figures 9 and 10 show the WNNT values (respectively for 5 MB-sized and 3 MB-sized songs) depending on the two following simulation traffic parameters:

- the speed at which users move throughout the cell (expressed in km/h),
- the additional traffic in the cell (expressed via erlang values).

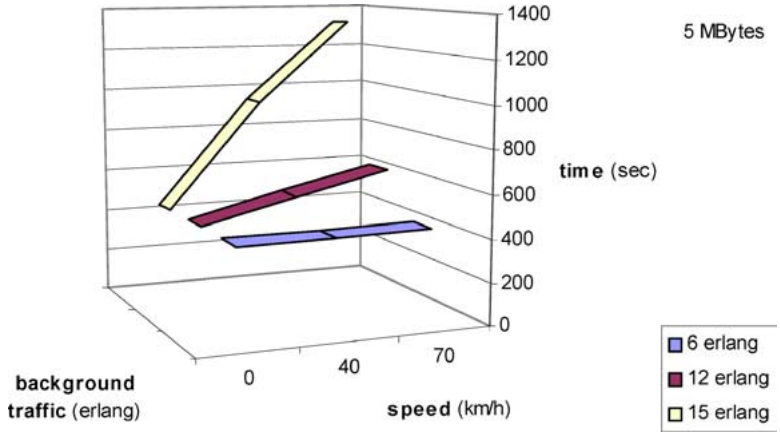

Figure 9. WNTT values for 5 MB-sized songs.

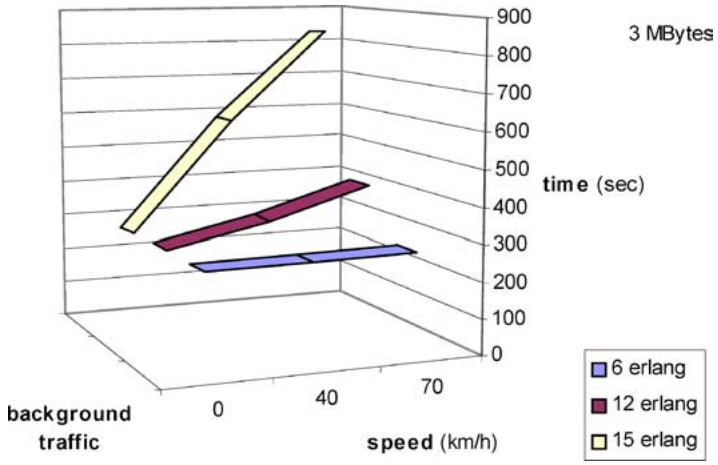

Figure 10. WNTT values for 3 MB-sized songs.

Two main considerations about the results of figures 9 and 10 are in order: (i) the larger the traffic in the cell (and the user speed), the larger the corresponding WNTT values, and (ii) the best WNTT result may be obtained when the mobile device is completely still. (In such a case a data rate of about 12 KB/s may be obtained.) Figure 11 summarizes the behaviour of the WNNT values for some different traffic combinations, depending on the song sizes. In particular from the top to the bottom of the graph represented in figure 11, the curves for the following traffic parameters have been respectively plotted: 15 erlang-70 km/h, 15 erlang-40 km/h, 12 erlang-70 km/h, 12 erlang-40 km/h, 6 erlang-0 km/h. It is easy to note that the more the song size increases (along with the amount of traffic in the cell) the more the WNNT values increase. As an example figure 12 reports

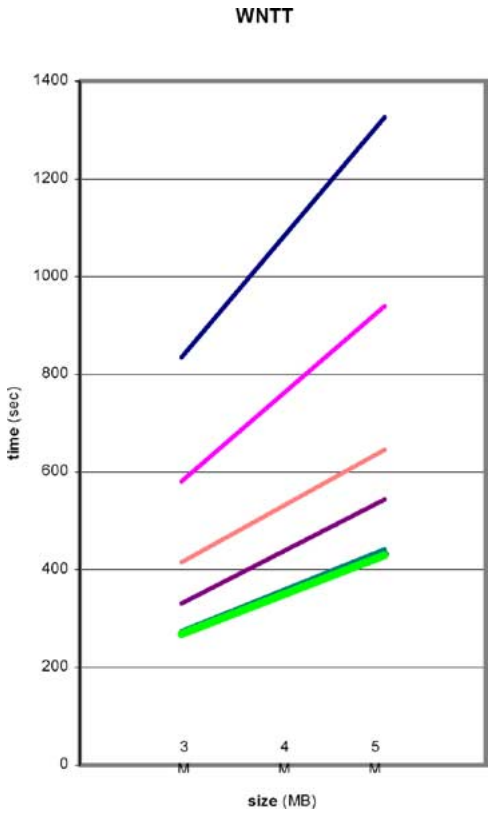

Figure 11. Summary of WNTT values depending on song size.

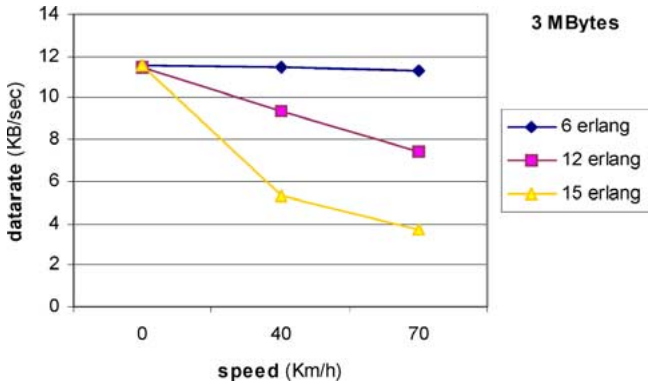

Figure 12. Data rate for 3 MB-sized songs.

on the average data rates that may be obtained on the wireless link for the download of songs of 3 MB (depending on the user speed). As expected, the larger the user speed, the smaller the obtained data rate. An important consideration is in order now, which is related to the impact that the download time values, obtained at the Internet side, have on the total time requested to download songs on UMTS terminals. The obtained average download delays at the Internet side (about 33 seconds) seem to be quite irrelevant if compared with the WNTT values which have been experienced on the wireless links (ranging from 250 to 1325 seconds, i.e. from about 4 to about 22 minutes). This optimal result at the Internet side is probably due to the use of the adopted Web replication technology along with the use of our distribution mechanism (C^2LD). Note, in fact, that if we try to download songs from a single Web server (such as the New Zealand Web server) with the standard HTTP, this can lead to an increase of the WNTT value by about 600 seconds (10 minutes).

To conclude this Subsection, it is worth mentioning that additional experiments were carried out to assess the ability of the session mechanism embodied in our architecture to provide reliable operations over unstable wireless links. To conduct those experiments we replaced the simulated UMTS layer of our wireless architecture by a real GPRS layer where real handoffs and long temporary link outages may be experienced.

In those experiments (about 30), in the midst of a song download session we entered with our mobile device an area of no signal coverage for a variable period of time ranging from several seconds (about 20) to several minutes (about 10). The result of this activity was the following. When the period of no signal coverage was limited within a maximum value of a very few minutes (around 1 or 2 minutes), the underlying TCP connection was able to respond to link restoration, even if a considerable amount of time (several seconds) passed from the precise instant the link became operational and the instant when TCP started packet retransmissions. Instead, when the routing level repaired the link outage after a longer period (about 3 minutes) the result was that the TCP connection was either destroyed or completely damaged, with no possibility to resume packet transmission at the TCP level. In these negative cases, our session mechanism was triggered and, after that the signal was obtained again at the link level, a new TCP connection was established to resume the data stream and to guarantee download continuity. The result of the activity of the session mechanism was that all the download operations were successfully completed, even though download times got unavoidably larger. The additional latencies that were introduced amounted to the time needed to repair the link.

3.3.3. Download times for sets of songs on UMTS links. It is well known that when a user wishes to listen to a set of songs, he would prefer to be able to listen to each single song in sequence, without any interruption occurring in between subsequent songs. Now, typical Mp3 songs with a 128 Kbit encoding (based on a 44100 Hz sampling rate) produce a needed data throughput of about 17 KB per second. Unfortunately, we already know from the simulation results presented earlier that, even in the best case (6 erlang of traffic in the cell, user speed equal to 0 km/h), the data rate which may be achieved does not exceed the value of 12 KB per second.

A possible solution to guarantee an uninterrupted playout amounts to keep the user waiting for a certain initial time period before he can begin to listen to the songs contained in the compilation.

Based on the WNTT results obtained for the download of single songs, we have tried to estimate the preliminary amount of time that a user should wait before he can begin to listen to the songs contained in a compilation without any interruption. The results of this theoretical estimation are reported in figures 13 and 14 for two given cases, respectively Case 1: song size = 4 MB, traffic = 6 erlang, user speed = 0 km/h; and Case 2: song size = 4 MB, traffic = 12 erlang, user speed = 70 km/h.

To validate the theoretical results presented in figures 13 and 14, we carried out an additional set of experimental trials by downloading sequences of different subsequent songs. Figures 15 and 16 show the WNTT results for compilations composed by 3 and

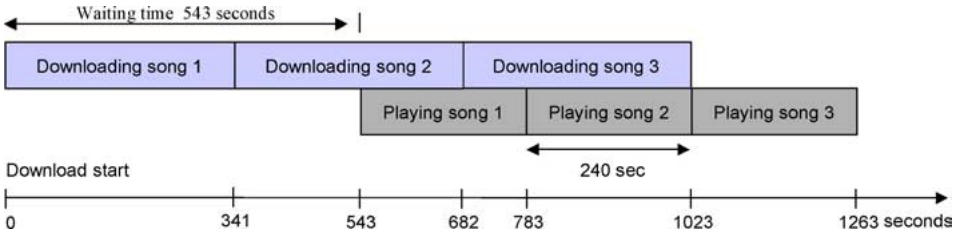

Figure 13. Waiting time for a 3-song compilation (song size: 4 MB, traffic: 6 erlang, user speed: 0 km/h).

Figure 14. Waiting time for a 3-song compilation (song size: 4 MB, traffic: 12 erlang, user speed: 70 km/h).

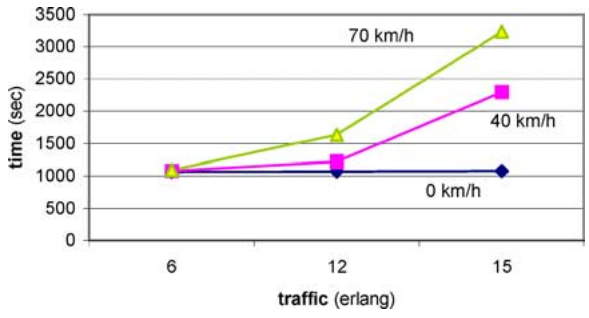

Figure 15. WNTT values for a 3-song compilation (12 MB).

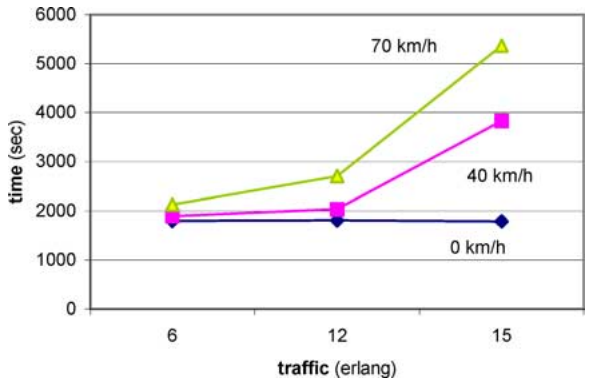

Figure 16. WNTT values for a 5-song compilation (about 19 MB).

5 songs, respectively. Each Mp3 song used in these experiments had a size of 4 MB. By analyzing the WNTT curves of figures 15 and 16, we may understand that the obtained experimental values are very close to the theoretical ones shown in the examples of figures 13 and 14. Hence, we may conclude that, based on an analysis of the curves of figures 15 and 16, one could derive quite precise estimates on the extent of time a user has to wait before he can start to listen to a compilation without any interruption.

4. Related work and comparison

The aim of this section is to briefly present some related work by other researcher and to compare it with the most relevant design choices we have taken to implement our system.

4.1. Content delivery networks

Some issues of paramount importance for the development of our system are those related to the problem of multimedia distribution over the Internet. In this context, in recent years, there has been plenty of emphasis about the possibility of an effective, secure and reliable access to multimedia information from mobile terminals. This has determined the evolution of architectural solutions and technologies based on *content*.

Whereas lower-layer network infrastructures are focused on the routing, forwarding, and switching of packets, the so-called (Internet-based) *content networks* deal with the routing and forwarding of requests and responses for *content* using upper-level application protocols. Typical data transported in content networks consist of images, movies and songs which are often very large in dimensions. According to a widespread definition, a Content Distribution Network (CDN) can be seen as a virtual network overlay of the Internet that distributes contents by exploiting multiple replicas. A request from a client for a single content item is directed to a *good* replica, where good means that the item is served to the client quickly compared to the time it would take if that item were fetched

from the original server. In the CDN-related literature it is said that a typical CDN has some combinations of a content-delivery infrastructure, a request-routing infrastructure and a distribution infrastructure.

The content-delivery infrastructure consists of a set of *surrogate* servers that deliver copies of content to users that issues requests for a certain content. The request-routing infrastructure consists of mechanisms that enable the connection of a given client with a selected surrogate. The distribution infrastructure consists of mechanisms that copy content from the origin server to the surrogates. Additionally, *content internetworking* allows different content networks to share resources so as to reach the most distant participants. A set of software architectural elements constitute the core of the Content Distribution Internetworking (CDI) infrastructure that use commonly defined protocols for content internetworking, as discussed at length in [3, 6, 8, 9, 19].

It is easy to recognize that the architecture of the wireless Internet application we have developed resembles the above mentioned CDI technology for the reason it interconnects a CDN, located in the Internet, with the UMTS network. At the basis of our CDI infrastructure we have put the Application Gateway which manages all the interactions between the UMTS terminals and the wired Internet. Our developed Intermediate System (IS), along with the set of all the replica servers, constitutes a real CDN: the content-delivery infrastructure is implemented by means of the replica servers which store multiple copies of Mp3 songs; the *search* functionalities of the Discovery, integrated with the C^2LD downloading mechanism that operates by engaging all the available replicas in supplying fragments of the required song, provide a combination of the request-routing infrastructure and the distribution infrastructure.

Another prominent issue in the design of our architecture for the distribution of musical contents is concerned with the fact that mobile clients have typically scarce computational capacity and need a stable access point to the wired CDN. Hence our choice to exploit a centralized entity (comprising the Application Gateway, the Download Manager and the Discovery) which functions like a stable and wired intermediary with respect to the decentralized musical resources.

To conclude this Subsection, it is worth noticing that while several CDNs have been created by a number of companies, there is little that has been published on the extent to which they are being used and their correspondent performance in serving contents [25, 28]. In this context, an interesting scheme called *Dynamic Parallel Access* (DPA) has been recently proposed by Rodriguez and Biersack that manages the distribution infrastructure in a CDN [40]. DPA exploits the parallelism inherent in globally replicated contents by involving all active replicas in the retrieval of a given resource. Unlike C^2LD, DPA splits a client's request into more sub-requests for resource portions which are of fixed size.

We have compared the download times which can be obtained by the C^2LD mechanism on the Internet side with those produced by an implementation of the DPA scheme. A simple CDN was used for our experiments with three different replica servers which were located in Europe: two of them were in Italy (Cesena and Trieste) and one in the UK. The client was running on a host in Bologna (Italy). 20 different experiments were carried out with files of different size, ranging from 0.1 KB to 11 MB.

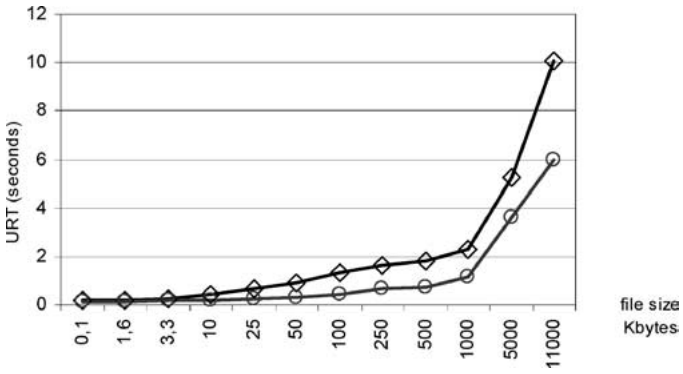

Figure 17. CDN performances: C^2LD vs. DPA.

The average results of our experimental comparison are plotted in figure 17 in terms of User Response Time (URT). As all the replica servers were quite near to the client, very small download times were experienced in all the experiments. As shown in figure 17, an important result of our experiments is that the C^2LD mechanism (lower curve) outperforms the DPA scheme (upper curve) when the size of the files exceeds 25 KB. Based on these results, we feel that C^2LD is currently one of the best candidate to implement the distribution infrastructure of a music delivery network.

4.2. Transport layer protocols

Mobile wireless is one of the more complex scenarios for Internet protocols, and in particular for transport protocols. One approach to supporting the wireless environment is the so-called *walled garden* approach [21]. According to this approach, the transport protocol used within the mobile wireless environment is not TCP, but is instead a transport protocol that has been specifically designed for mobile wireless. The WAP approach is exactly based on this kind of solution: almost every layer of the standard TCP-IP protocol stack has been redefined and modified, with the result of losing a full compatibility with standard IP applications which are designed for wireline environments [47].

An alternative solution to the walled garden approach is represented by the ALL-IP approach where mobile wireless devices are allowed to perform like any other Internet-connected device. If this approach is chosen, then some form of end-to-end TCP continuity is required to ensure that TCP operates efficiently on the wireless link. All the following events may occur on the wireless links which are interpreted by the standard TCP as signs of network congestion:

- communication pauses due to handoffs between cells,
- packet losses experienced when the mobile host enters a region of no signal coverage,
- packet losses due to transmission errors at the radio link level.

To react to these negative situations the standard TCP triggers exponential back off policies to achieve connection stability. Unfortunately, as both packet losses and delays are caused by host mobility (and not by congestion), performing such conservative schemes over wireless links results in further performance degradation, instead of relieving the connection.

As already mentioned, the quest to solve the performance problems encountered by standard TCP over wireless links is an active area of research, and several alternative solutions have been proposed [1, 2, 4, 5, 10, 12, 17, 20, 35, 42, 48, 49]. Some of those proposals have been of particular interest for our work. For the sake of clarity, we can classify many of the different proposed protocols into three categories [11]: split connection protocols [1, 20, 49], link layer protocols [2, 35] and end-to-end protocols [5].

The main idea behind the split connection approach is to split into two different parts the communication, with the base station in the middle, and to shift the bulk of network protocol from simplified mobile machines to the base station. In this way, the wired half of the connection does not require any changes, while a specialized protocol can be used at the wireless segment to improve performance.

A significant example of this approach is the *Indirect TCP for Mobile Hosts* (I-TCP) protocol [1]. I-TCP splits a standard TCP connection into two different parts, with the base station at the middle point. Then, modifications of the TCP protocol are proposed for the wireless communication segment which, exploiting the use of selective ACKs, permit to recover more than one lost TCP packet in one round trip time, thus improving the final throughput [49].

Also in [20] a TCP connection is split into two parts at the base station (*Mobile TCP*). The designers of Mobile TCP devise a simplified communication mechanism which can replace the standard TCP on the wireless half of the connection. Based on the consideration that a base station is more powerful and stable than a mobile device, they move to the base station much of the work due to the connection management and the flow control of the wireless link.

The link layer approach, instead, tries to hide the characteristics of the wireless link from the transport layer and solves performance problems at the link layer.

As an example, in [2] Balakrishnan et al. propose the use of the SNOOP module between the TCP and the IP layers. The idea of SNOOP is to monitor, at the base station, both the packet addressed to the mobile hosts and the relative acknowledgements sent by the mobile host. Packets not yet acknowledged by the mobile host are buffered by the base station. If a packet loss is detected, the base station can directly exploit buffered packets to manage retransmissions. In the scheme proposed by the designers of SNOOP, the problem of managing delays due to handoffs and temporary link outages is also faced by using a complex mechanism based on buffering and multicast transmissions.

Another link-layer oriented approach is presented in [35], where Parsa and Garcia-Luna-Aceves present a link improvement protocol (*TULIP*) that allow TCP to operate efficiently in the face of the unstable conditions of wireless links. The principal architectural characteristic of this approach is that it proposes a MAC acceleration feature which can be applied to collision avoidance mechanisms (e.g., IEEE 802.11) to improve throughput over wireless lossy links.

Finally, the most studied category is that of end-to-end protocols. Here, researchers attempt to handle losses an delays in a way that improve the performance of TCP over wireless links, while maintaining the end-to-end semantics of regular TCP.

A very significant example of this approach is provided in [5]. In this paper, Caceres and Iftode propose very practical solutions to improve the TCP performance on a wireless link. The first solution amounts to implement a sort of smooth, "make then break" handoff procedure in order to eliminate packet losses during cell crossings. The second type of solution is identified in the necessity to engineer more accurate retransmission timers, while a final and more attractive solution is to resume communications immediately after that handoffs complete, without waiting for retransmission timeouts (fast retransmissions).

Also there exist proposed protocols which rest upon a combination of the assumptions taken by the three different categories we have mentioned [4, 10, 12, 17, 42, 48, 49]. For example, in [48] Wang and Tripathi propose a new protocol that replaces the standard TCP/IP protocol over the wireless link by a simpler protocol with smaller headers. In addition, in their scheme several functions needed for the communication between a mobile host and the Internet are shifted to the base station, and, finally, link-layer acknowledgements and retransmissions are exploited to quickly recover losses over the wireless link.

Along the line of quick reactions to link layer problems, there is a recent paper by Goff, Moronski and Phatak where the *Freeze-TCP* model is presented [17]. In this paper, the authors propose a pro-active action/signaling method that may be employed by mobile hosts to react to disconnections with a higher performance than standard TCP.

Summarizing, the split connection approach has several important advantages. The first is that the mobile machines may be hidden from outside, so they can be specialized and simplified. Secondly, it is a good idea to delegate the most complex part of networking tasks to base stations as they are, typically, more powerful and stable than mobile devices. However a typical drawback is that most part of the proposed schemes require intermediaries where the TCP traffic is monitored and controlled (e.g., base stations).

Shifting these functionalities from mobile hosts to intermediaries may give rise to scalability problems. For example, a base station may become overloaded if it has to serve a large amount of mobile devices with multiple connections.

An even more important drawback, which is common to almost all the proposed protocols, is that those protocols require (from significant to modest) changes in the TCP protocol stacks. In the worst cases, those changes may be required on the sender side or on the intermediate machines. It is better when changes in the TCP code are restricted to the mobile client side, making it possible to interoperate with the existing infrastructure.

To conclude this Subsection, it is worth reminding that a problem of practical relevance for music distribution to mobile devices is that of an unexpected link interruption in the midst of a long song download activity. To surmount this type of problems, we have equipped our architecture with a wireless *session layer* developed on the top of the standard TCP protocol. The aim of our session level is to ensure that the song download activity is not interrupted when the data link level communication is destroyed due to very long link outages or handoffs. In essence, our session mechanism was not designed to solve the problem of adjusting TCP performance to match the requirements of the wireless environment. Instead, our session was developed to guarantee a successful termination of the download

activity even when the underlying TCP connection is disrupted due to the device mobility. Based on these considerations, it is possible to envisage that our session mechanism may be successfully "mounted" on top of one of the TCP wireless protocols which were proposed to alleviate the worst aspects of performance degradation of the standard TCP protocol.

4.3. Wireless access to Web contents

In figure 18 we illustrate three alternative methods to access Web contents from mobile devices. In the rightmost side of figure 18, the approach is illustrated which exploits a specific wireless protocol to access Web contents from mobile devices. A protocol gateway uses this specific wireless protocol to enable the interaction of the wireless device with the Internet. An already mentioned example of such type of solution is the WAP approach that incorporates a protocol gateway able to translate requests from the wireless protocol stack to the Web protocols. Moreover, instead of using HTML, WAP uses the Wireless Markup Language (WML), a subset of XML, to create and deliver contents [47].

With regard to this issue, it is important to notice that the IS embodied in our architecture performs different functions with respect to the protocol gateway of the WAP solution (see figure 18). The WAP-type gateway performs translations from HTML-based contents to the proprietary format which is understandable at the mobile terminal, instead our proposed Application Gateway does not performs any form of content translation, but only implements a Content Delivery Internetworking infrastructure.

There exist also simpler microbrowser architectures (such as, for example, Microsoft's Mobile Explorer) that completely rest on the use of the HTML markup language for content delivering. However, a great limitation of this approach depends on the server ability to send information in pure HTML (leftmost part of figure 18) [41]. Besides WAP, microbrowser technology still moves forward with innovative solutions such as, for example, iMode and the Pixo Internet Microbrowser. Those protocols are specifically aimed at the wireless

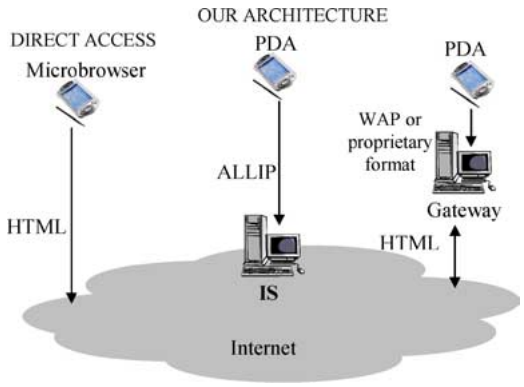

Figure 18. Wireless access to web contents: microbrowser, music-on-demand, WAP.

Internet, since they re-code the Internet content for wireless devices and utilize Compact HTML (CHTML) or Extensible Hypertext Markup Language (XHTML) as their markup languages [27, 29, 41].

Middleware often offers an alternative to manually replicating content. Its basic purpose is to transparently transcode content *on the fly* without maintaining Web content in multiple formats. The use of the Relational Markup Language (RML), the Parlay project [45], the micro edition of Javag (J2ME) [23], the Mobile Execution Environment (MExE) [30], the micro version of Jini (JMatos) [24], Online Anywhere [34] and Proxynet [37] are all initiatives which fall inside the middleware-based approach.

It is not possible to conclude this Subsection without any mention to the JXTA technology [18, 36]. This is a set of open (peer-to-peer) protocols that allow any connected device on the network (from cell-phone to PDA, from desktop computer to consumer electronics) to communicate. The focus of JXTA protocols is on creating a virtual network overlay on top of the Internet allowing peers to directly interact independently of their network location, programming language, and different implementations. At the heart of JXTA technology we can find *advertisements* (XML documents) that are exploited to advertise all network resources (from peers to contents). Advertisements are exploited to provide a uniform way to publish and discover network resources.

5. Conclusions

We have developed a wireless Internet application designed to support music distribution to UMTS devices with the Internet as a backplane. Simply stated, our software architecture has been designed to interconnect a Music Delivery Network, located in the Internet, with UMTS-based clients. From a final-user standpoint, our application enables mobile clients to download and to listen to Mp3 files on UMTS devices.

We reported on a large set of experimental results we obtained on the field by exploiting our wireless application. The download time measurements we have experimentally obtained show that combining 3G mobile network technologies with an appropriate structuring of the wireless Internet application may be very effective for the *fast* distribution of music to mobile clients. We conclude this paper by noticing that the WNTT values we have obtained from our experiments are in the range from 250 to 1325 seconds, for single songs, depending on the user's speed and on the additional traffic present in the cell. With different wireless technologies, and in the absence of additional traffic, we would have obtained theoretical WNTT values ranging from about 1500 (GPRS technology at 28.8 Kb/s) to around 3000 seconds (GSM technology at 14.4 Kb/s) [38, 39].

In the near future, we expect to be able to assess the software architecture, which is at the basis of our wireless application, by using a real UMTS network infrastructure. In addition, future research efforts will be devoted to the activity of porting our Internet wireless application on different operating platforms. We wish to mention that currently: (i) we have not addressed yet any security and copyright protection issues for our application, (ii) we have not considered yet the design of an accounting infrastructure. We plan to devote our future research to these relevant issues.

Acknowledgments

This research was conducted with the financial support from the EU, the Italian MIUR, the "Fondazione Marconi of Bologna", the Department of Computer Science of the University of Bologna and Microsoft Research Europe. We are indebted to our colleagues Stefano Cacciaguerra, Alessandro Gambetti, Davide Melandri, Orlando Orlandi, Mirko Piaggesi and Daniela Salsi for their helpful assistance during the implementation and testing of the software architecture discussed in this paper. Finally, we are grateful to the anonymous referees of the *International Journal on Multimedia Tools and Applications* for their helpful review of this article.

References

1. A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in Proc. International Conference on Distributed Computing Systems, Vancouver, Canada, May 1995.
2. H. Balakrishnan, S. Seshan, E. Amir, and R.H. Katz, "Improving TCP/IP performance over wireless networks," in Proc. of Mobicom '95, Berkeley, California, USA, Nov. 1995.
3. A. Barbir, B. Cain, F. Douglis, M. Green, M. Hoffmann, R. Nair, D. Potter, and O. Spatscheck, "Known CDN request-routing mechanisms," draft-cain-cdnp-known-request-routing-02.txt, June 2001.
4. P. Bhagwat, P. Bhattacharya, A. Krishna, and K. Tripathi, "Using channel state dependent packet scheduling to improve TCP throughput over wireless LANs," Wireless Networks, Vol. 3, No. 1, 1997.
5. R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," IEEE Journal on Selected Areas in Communications, Vol. 13, No. 5, pp. 850–857, 1995.
6. B. Cain, O. Spatscheck, M. May, and A. Barbir, "Request-routing requirements for content internetworking," draft-cain-request-routing-req-02.txt, July 2001.
7. M. Conti, E. Gregori, and F. Panzieri, "QoS-based architectures for geographically replicated web servers," Cluster Computing, Vol. 4, pp. 105–116, 2001.
8. M. Day, B. Cain, G. Tomlinson, and P. Rzewski," A model for content internetworking," draft-day-cdnp-model-05.txt, March 2001.
9. M. Day, D. Gilletti, and P. Rzewskip, "CDN peering scenarios," draft-day-cdnp-scenarios-03.txt, March 2001.
10. D.A. Eckhardt and P. Steenkiste, "Improving wireless LAN performance via adaptive local error control," in Proc. Sixth IEEE International Conference on Network Protocols, Austin, Texas, USA, Oct. 1998.
11. H. Elaarag, "ImprovingTCP performance over mobile networks," ACM Computing Surveys, Vol. 34, No. 3, pp. 357–374, 2002.
12. A. Fieger and M. Zitterbart, "Evaluation of migration support for indirect transport protocols," Proc. Second Global Internet Conference, Phoenix, Arizona, USA, Nov. 1997.
13. A. Forouzan, TCP/IP Protocol Suite, McGraw Hill: New York, 2000.
14. Freenet Project Inc., The Freenet project, http://freenet.sourceforge.net
15. V. Ghini, F. Panzieri, and M. Roccetti, "Client-centered load distribution: A mechanism for constructing responsive web services," in Proc. 34th International Conference on System Sciences, Maui, Hawaii, USA, Jan. 2001.
16. Gnutella official site, http://gnutella.wego.com/
17. T. Goff, J. Moronski, D. Phatak, and V. Gupta, "Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments," in Proc. of IEEE Infocom 2000, Tel-Aviv, Israel, March 2000, pp. 1537–1545.
18. L. Gong, "JXTA: A network programming environment," IEEE Internet Computing, May-June 2001, pp. 88–95.
19. M. Green, B. Cain, G. Tomlinson, S. Thomas, and P. Rzewskip, "Content internetworking architectural overview," draft-green-cdnp-gen-arch-03.txt, March 2001.

20. Z. Haas and P. Agrawal, "Mobile-TCP: An asymmetric transport protocol design for mobile systems," in Proc. International Conference on Computers and Communications, Montreal, Canada, June 1997, pp. 1054–1058.
21. G. Huston, "TCP in a wireless world," IEEE Internet Computing, March–April 2001, pp. 82–84.
22. D. Ingham, S.K. Shrivastava, and F. Panzieri, "Constructing dependable web services," IEEE Internet Computing, Vol. 4, No. 1, pp. 25–33, Jan./Feb. 2000.
23. JAVA J2ME, http://www.java.sun.com/j2me
24. JINI Network Technology, http://www.sun.com/jini/index.html
25. K.L. Johnson, J.F. Carr, M.S. Day, and M.F. Kaashoek, "The measured performance of content distribution networks," in Proc. Fifth International Web Caching and Content Delivery Workshop, Lisbon, Portugal, May 2000.
26. R. Kalden, I. Meirick, and M. Meyer, "Wireless internet access based on GPRS," IEEE Personal Communications, Vol. 7, No. 2, pp. 8–18, April 2000.
27. T. Kanter, "An open service architecture for adaptive personal mobile communication," IEEE Personal Communications, Vol. 8, No. 6, pp. 8–17, 2001.
28. B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of content distribution networks," in Proc. ACM SIGCOMM Internet Measurement Workshop, San Francisco, California, USA, 2001.
29. G. Lawton, "Browsing the mobile internet," IEEE Computer, Vol. 34, No. 12, pp. 18–21, 2001.
30. MexE Forum, http://www.mexeforum.org
31. Microsoft, Windows CE home page, http://www.microsoft.com/windows/embedded/ce/default.asp
32. MP3 resources by MPEG.ORG, http://www.mpeg.org/MPEG/mp3.html
33. Napster official site, http://www.napster.com/
34. Online Anywhere, http://www.onlineanywhere.com/
35. Parsa and J. Garcia-Luna-Aceves, "Improving TCP performance over wireless networks at the link layer," ACM Mobile Networks and Applications Journal, Vol. 5, pp. 57–71, April 2000.
36. Project JXTA , http://www.jxta.org
37. Proxinet, http://www.pumatech.com/proxinet
38. M. Roccetti, V. Ghini, and P. Salomoni, "Distributing music from IP networks to UMTS terminals: An experimental study," in Proc. 2002 SCS Euromedia Conference, Modena, Italy, April 2002, 147–154.
39. M. Roccetti, V. Ghini, P. Salomoni, A. Gambetti, D. Melandri, M. Piaggesi, and D. Salsi. "The structuring of a wireless internet application for a music-on-demand service on UMTS devices," in Proc. ACM Symposium on Applied Computing, Madrid, March 2002, pp. 1066–1073.
40. P. Rodriguez and E.W. Biersack, "Dynamic parallel access to replicated contents in the internet," IEEE/ACM Transactions on Networking, Aug. 2002
41. S. Saha, M. Jamtgaard, and J. Villasenor, "Bringing the Wireless Internet to Mobile Devices," IEEE Computer, Vol. 34, No. 6, pp. 54–58, June 2001.
42. N. Samaraweera and G. Faerhurst, "Reinforcement of TCP error recovery for wireless communications," ACM SIGCOMM, Computer Communication Review, Vol. 28, No. 2, pp. 30–38, 1998.
43. W.R. Stevens, TCP/IP Illustrated, Vol. I, Addison Wesley: Reading, Massachusetts, USA, 1998.
44. D. Staehle, K. Leibnitz, and K. Tsipotis, "QoS of internet access with GPRS," in Proc. Fourth ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Rome, Italy, July 2001, pp. 57–64
45. The Parlay Group, www.parlay.org
46. UMTS Forum, "What is UMTS?," http://www.umts-forum.org/what_is_umts.html
47. Wap Forum, "WAP Architecture Specification," http://www1.wapforum.org/tech/terms.asp?doc=WAP-100-WAPArch-19980430-a.pdf
48. K. Wang and S.K. Tripathi, "Mobile-end transport protocols: An alternative to TCP / IP over wireless links" in Proc. IEEE Infocom '98, San Francisco, California, USA, April 1998, pp. 1046–1053.
49. R. Yavatkar and N. Bhagawat, "Improving end-to-end performance of TCP over mobile internetworks," in Proc. International Workshop in Mobile Computing Systems and Applications, Santa Cruz, California, USA, Dec. 1994.

Marco Roccetti is a Professor of Computer Science at the Department of Computer Science of the University of Bologna. From 1992 to 1998 he was a research associate at the Department of Computer Science of the University of Bologna, and from 1998 to 2000 he was an Associate Professor of Computer Science at the University of Bologna. Marco Roccetti authored 70+ technical refereed papers that appeared in the proceedings of several international conferences and journals. His research interests include: protocol design, implementation and evaluation of wired/wireless multimedia systems, performance modeling and simulation of multimedia systems, digital audio for multimedia communications.

Paola Salomoni is an Associate Professor of Computer Science at the Department of Computer Science of the University of Bologna. In 1992 she received the Italian Laurea degree (with honors) in Computer Science from the University of Bologna. From 1995 to 2001 she was a research associate at the Department of Computer Science in Bologna. Her research interests include the following: design and implementation of distributed multimedia systems, integration of multimedia services in wireless networks, design and implementation of teaching/learning environments.

Vittorio Ghini is an Assistant Professor of Computer Science at the Department of Computer Science of the University of Bologna. He received the Laurea degree (with honors) and the Ph.D. in Computer Science from the University of Bologna respectively in 1997 and in 2002. His current research interests include QoS management at the middleware level, Web performances, network emulation and architectural design of Internet based multimedia systems.

Stefano Ferretti received the Italian Laurea degree with honors in Computer Science from the University of Bologna. Actually he is a Ph.D. student in Computer Science at Computer Science Department of the University of Bologna. His research interests are related to the design, implementation and evaluation of wired/wireless distributed multimedia systems.