

Managing Application Level Quality of Service through TOMTEN

R. De Silva **, B.Landfeldt*, S.Ardon*, A. Seneviratne*, Christophe Diot***

[ranil, bjornl, seb, aps]@eng.uts.edu.au

* School of Electrical Engineering, **School of Computing Science

University of Technology, Sydney, PO Box 123 Broadway,

Ultimo NSW 2007, Australia

cdiot@sophia.inria.fr

*** INRIA, 2004 route des Lucioles, B.P. 93, 06902 Sophia Antipolis, France

Keywords: QoS management, adaptive applications, and configurable protocols

Abstract

With the growth of mobile computing, users will have simultaneous access to multiple overlaid networks, each with different characteristics, services and costs. This paper introduces TOMTEN (TOtal Management Of Transmissions for the ENd-user), a framework for managing resources in both this type of environment as well as the traditional single network environment. TOMTEN is a reactive framework that is only activated when either the application is started or the user indicates dissatisfaction with a current session. The reactive architecture of TOMTEN also supports application adaptivity. Application adaptivity in TOMTEN does not require modifications to the application and is done transparently using proxy modules.

1. Introduction

With the growth of mobile computing, users will have access to heterogeneous networks consisting of both wired and wireless networks. It is likely that these networks will be overlaid thereby allowing the user to have simultaneous access to more than one network. This environment will be characterised by variable services offered by both the end-system and the different networks it is attached to. The variations will be primarily characterised by changes in available power, CPU capacity, bandwidth and cost. This paper introduces TOMTEN (TOtal Management of Transmissions for the ENd-user), a framework for managing resources in such environments to maximise the benefit to the user, i.e. maximise the quality of service.

As perception of quality is finally dependent on the user, it is necessary to assist the user in controlling his/her operational environment as s/he sees fit, to get the level of service required [10]. This enables the users to react to the state of the system rather than having the system decide on acceptable levels of service that have been pre-specified. Therefore, a reactive framework does not require constant monitoring, determination of thresholds for various system resources to trigger changes. This significantly simplifies the introduction of QoS management functionality compared to using a predictive approach used in the Quality of Service (QoS) management frameworks to date [13].

TOMTEN is based on the above reactive paradigm. It does not make decisions about the acceptability of a session. This decision is left to the user and TOMTEN provides the infrastructure to gather and present information that will help the user to make decisions, and the necessary controls to enforce the decision. Thus, TOMTEN acts as an agent between the user and the system, through which the available resources can be manipulated.

In Section 2, we will introduce the three components that constitute TOMTEN, – USA, PRIMATE and Boomerang. In Section 3, we discuss how these components interact within TOMTEN. Section 4 describes a prototype implementation of the framework, and presents experimental results from using two applications with TOMTEN. Finally, section 5 presents the conclusions.

2. TOMTEN

As mentioned, the aim of the framework is to maximise the user satisfaction through the active management of available resources. We try to achieve this goal by supporting a highly flexible architecture, and simplifying the use of it.

TOMTEN provides flexibility in three dimensions: the application, the communication sub-system and the network. Application flexibility is provided by allowing users to alter an application's mode of operation on the fly. This is done by placing service agents acting as proxies, in-between the application and the end-system. These agents provide the adaptation functionality similar to that provided by the distillers described by [8], however as it is done external to the application, use of the agent does not require any modifications to the applications.

Communication flexibility is provided using application and system specific protocol stacks [4]. Network flexibility is provided by allowing different networks to be used at different times during a session, as well as managing resource reservations where possible, thus enabling the users to make cost-performance trade-offs.

This is achieved using the three modules as shown in Figure 1:

- A reactive quality of service management module called USA (User Services Assistant) [10], that monitors the system state if the user indicates that s/he is unhappy, and provides suggestions as to what the can be done,
- An application specific adaptation module called PRIMATE (PROxy Intelligent Module for Adapting Traffic Efficiently) that provides the adaptation functionality to the applications,
- Boomerang, a module which intercepts the traffic from the application and redirects it to the PRIMATE, thereby eliminating the need to modify applications to be used with the TOMTEN framework, and

The following sections explain these components in more detail.

2.1 USA

USA (User Services Assistant) is the reactive quality of service manager at the centre of the TOMTEN framework. As mentioned earlier, most QoS management schemes, that have been proposed to date, actively monitor applications and tries to keep the resource usage within certain boundaries. If the monitoring processes detect QoS violations, the system attempts to compensate by re-negotiating local and network resources. These re-negotiations are pre-specified, and are done without user intervention or knowledge in the “belief”, that they will improve the perceived level of quality.

The perception of quality is dependent on a number of factors like the user's individual preferences, human factors such as impaired vision, the content of the media stream and the operating environment such as ambient noise. Thus, we believe it is not possible for the system to predict the user's perception of quality and use pre-specified re-negotiation options. Therefore, unlike other QoS managers, USA is *reactive*, and is only activated at the start of a session and during a session, when the users indicates that they are dissatisfied with the current level of quality.

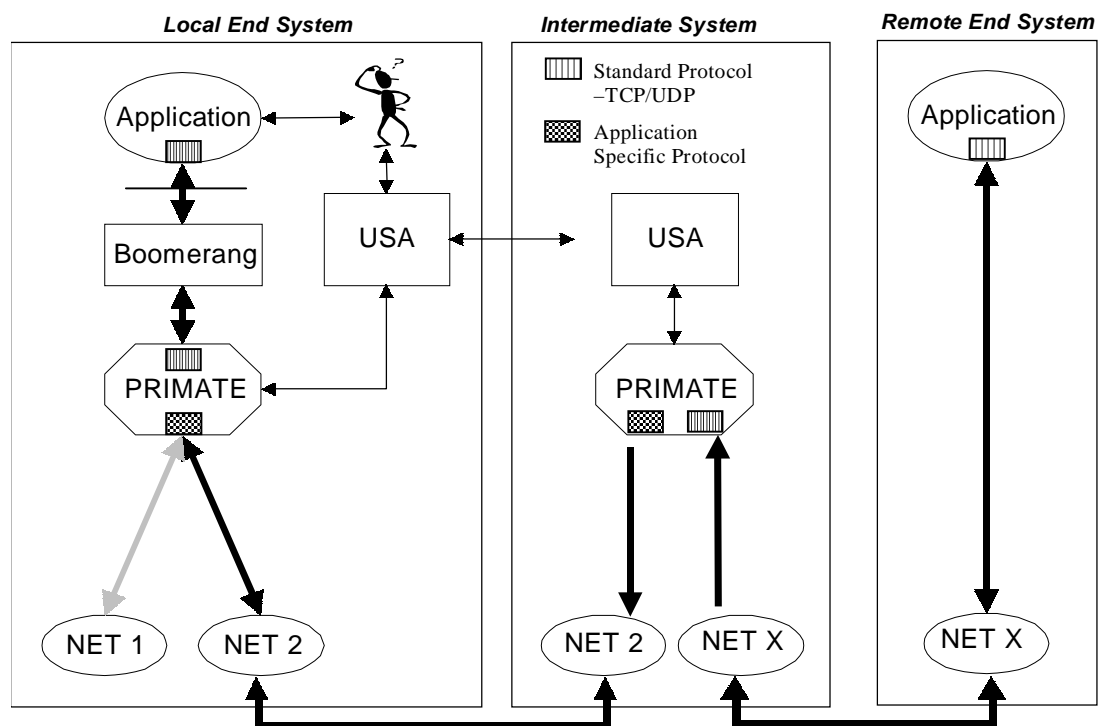


Figure 1. The TOMTEN Framework

Thus, USA does neither continuously monitor the system nor automatically change the negotiated resources levels. Instead, when activated by the user, it probes the system to determine its state and suggests a possible set of actions that may improve performance. The user selects his/her choice, and USA configures the system accordingly. If the suggested changes are unacceptable or unable to improve the application performance, the user has to restart the session with a new set of parameters. The user therefore makes conscious decisions about the system, e.g. new costs of the connection – something schemes that automatically re-negotiate resources fail to do.

Figure 2 shows the different modules that constitute USA. Given that the user is an integral part of this framework, it is vital that the user can easily interact with USA. This is achieved using a simple Graphical User Interface (GUI), a button, which the user presses to indicate dissatisfaction.

When the user presses the button, USA uses resource probes to determine the changes in system resources that occurred since it was last activated. The changes in the resource levels can be used to infer the cause of user dissatisfaction as described in [3]. USA is not dependent on any specific resource probing tools. Furthermore, it can operate with only a limited number of resource probes. However, as the system state is used by USA to determine the most appropriate course of action, the quality of the system state information will influence USA's effectiveness. In addition to resource probes, USA also supports resource managers, which provide the necessary functionality where possible, to reserve host computer system and network resources. This enables USA to handle different resource reservation procedures, e.g. in the case of networks, for both ATM and RSVP.

The PRIMATE manager in USA maintains information of the installed PRIMATE modules, and how they are to be used. The PRIMATE manager is also responsible for downloading new PRIMATE modules and running them on demand, as described in section 2.2 below. The logic module contains the logic that will enable the computation of the possible adaptation options when requested by the user. The MIB stores information about the application, this information is optional, and may be entered when the application is registered with TOMTEN. The MIB information is used to facilitate the decision making process, within USA.

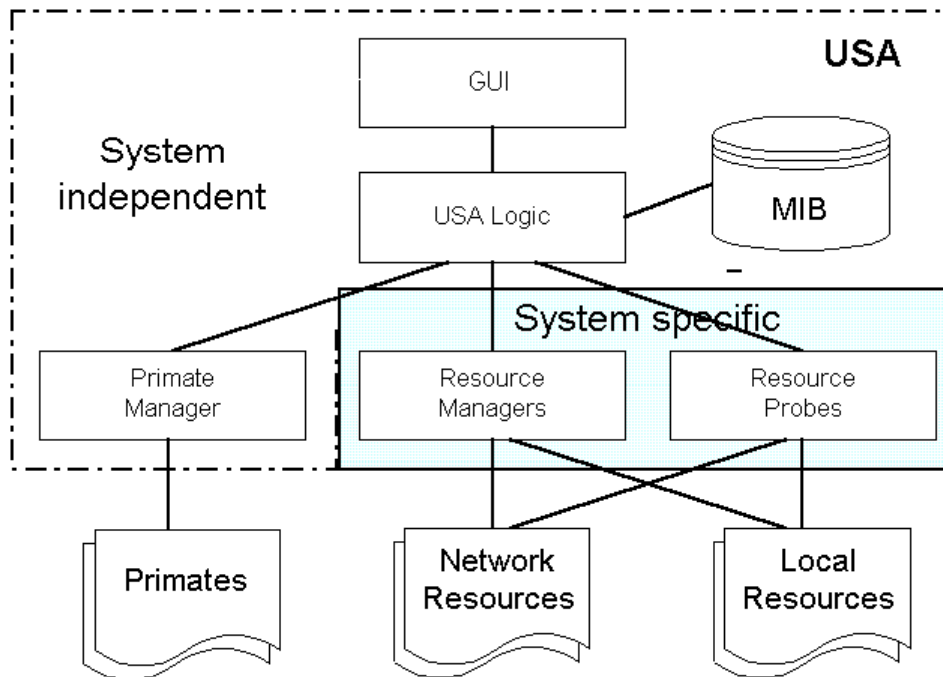


Figure 2 USA - Reactive Quality of Service Manager

2.2 PRIMATE

The PRIMATE (PROxy Intelligent Module for Adapting Traffic Efficiently) provides application adaptation functionality within the TOMTEN framework. Application adaptivity is traditionally built into the application [6]. Consequently, each application attempts to monitor the system state, and adapt its behaviour accordingly. This approach has two major shortcomings. Firstly, as the monitoring is done within the application it does not have a global view of the system state. This can lead to system instability as shown by [7]. Secondly, the requirement for monitoring and adaptation functionality significantly increases the complexity of the applications. TOMTEN overcomes this by separating application adaptivity into an independent module, namely a PRIMATE. Application data streams are then simply intercepted and redirected to it as described in Section 2.3. In addition, the PRIMATES use the global monitoring functionality offered by USA. Adaptation is thus provided by PRIMATES, and dynamically controlled by the user through USA.

PRIMATE modules are content specific, rather than application specific. Therefore, once developed can have wide applicability. For example, a PRIMATE developed to provide adaptivity for MPEG data could be used with any MPEG application. Moreover, as PRIMATES are developed separately to the application, they do not require access to application source, and encourage development by third party vendors.

Figure 3 shows the basic components of PRIMATE. The services a PRIMATE offers are dependent firstly on content specific knowledge, and where appropriate, on a tailored communication subsystem. The content specific knowledge provides information about how the application data stream can be adapted. The application content itself can consist of a number of media types. For example, a web browser that can support different data types (i.e. different image formats, audio or video coding schemes). In these cases, the content knowledge is used to configure a number of media-specific adaptation modules as shown in Figure 3.

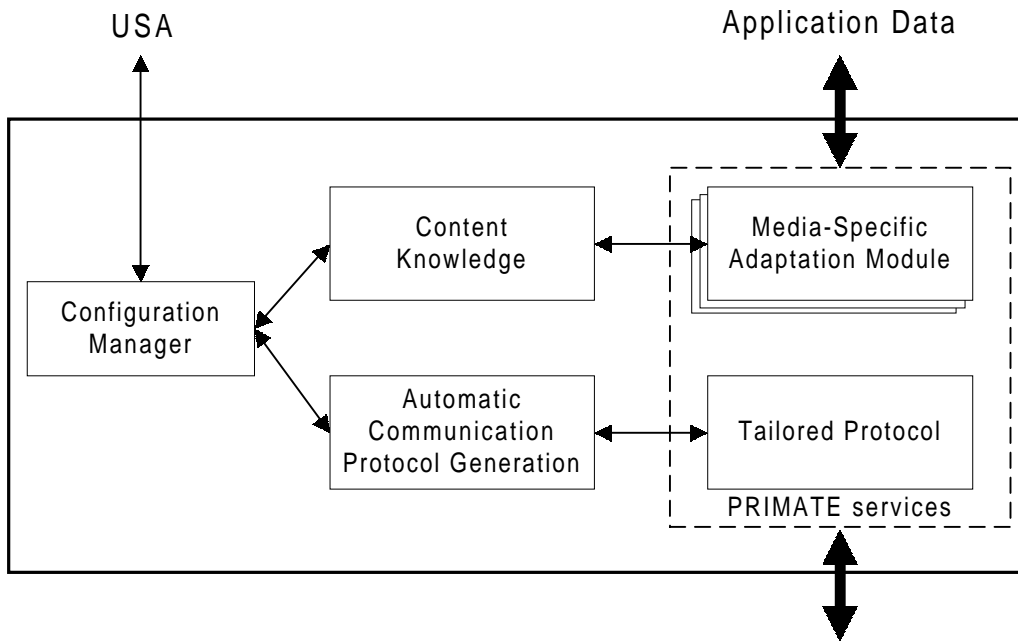


Figure 3 PRIMATE

The second component of PRIMATE's services is based on the use of a tailored communication protocol stack. A tailored communication protocol stacks results in a communication subsystem capable of specifically supporting the application's requirements and providing functionality specific to the networking environment. It has been shown in [4] that these protocols can be dynamically generated using automated techniques, to improve application performance. PRIMATE's distributed design, allows tailored communication protocols to be easily supported between the local and remote components, without modification to either of the end applications. Thus, PRIMATE's protocol supports the development of indirect protocols as proposed by [1].

The final element of the PRIMATE module is the configuration manager that generates possible configuration suggestions. These suggestions are based on the content knowledge and possible configuration of communication subsystem. The suggestions are then returned to USA, which presents them to the user. When the user selects an option, USA sends the selection to the configuration manager, which reconfigures the service provided by the PRIMATE.

In addition, the PRIMATE can be used handle vertical handoff in mobile computing environments that will support multiple overlay networks [9]. The PRIMATE will retain session information, and when switching to another network, will simply tear down the existing connection and establish a new connection. The session information will also facilitate continuing the session if the connection is lost and re-established.

2.3 Boomerang

Boomerang is responsible for transparently intercepting the data transmitted by the application and redirecting it to the PRIMATE. This eliminates the need to modify the application to transfer data directly to PRIMATE when they do not support proxy processing (as done in some of the newer Internet applications). Boomerang provides a uniform solution to this problem as modifying applications is impractical and not all applications support proxy processing. Our implementation of Boomerang does not require any changes to the kernel. It is implemented as a new kernel module. Figure 4 illustrates how Boomerang will intercept traffic from the application in the implementation framework of TOMTEN, which is discussed in Section 4.1.

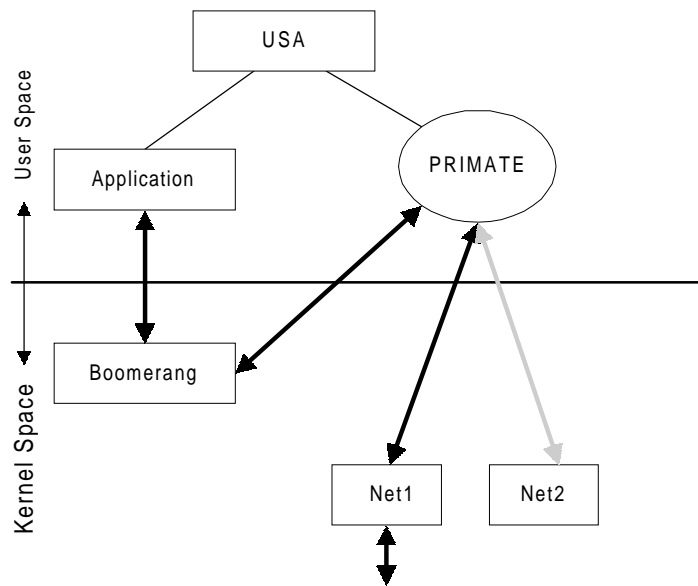


Figure 4 TOMTEN Prototype Implementation Architecture

3. The TOMTEN Framework

The user starts an application through USA's GUI, interaction (1) as shown in Figure 5. Before the application is launched, USA also gives the opportunity for the user to indicate the importance of the application, for example high, medium or low preference. Once the application's importance is known, USA checks the availability of the local and network resources (2) before suggesting a short list of the configurations that will suit the current operating environment (3). These suggestions are based on information obtained through the USA's MIB, and by probing the relevant system resources. The objective would be to, where possible, provide the user a choice of few (two or possibly three) suitable configurations, and leave the final decision to the user. The user has the choice of accepting one of the recommended configurations or renegotiating which a different set of parameters. For example, change the level of importance of the application so that USA will provide a new set of choices.¹

When the user makes a choice, USA configures the different TOMTEN components (4). This will involve notifying boomerang, and starting the appropriate PRIMATES. In addition, where possible, USA will make the necessary resource reservations. USA also contacts its peer entity on the intermediate host and requests it to start the remote PRIMATE (5).

Currently to simplify the TOMTEN model, we assume that the Intermediate system is always able to support any requests made. If this is not the case, TOMTEN will work in a "transparent mode". In the transparent mode, USA is the only active part of the framework. Finally, the application is started (6) by USA.

¹ We believe this will finally have to be based on monetary cost.

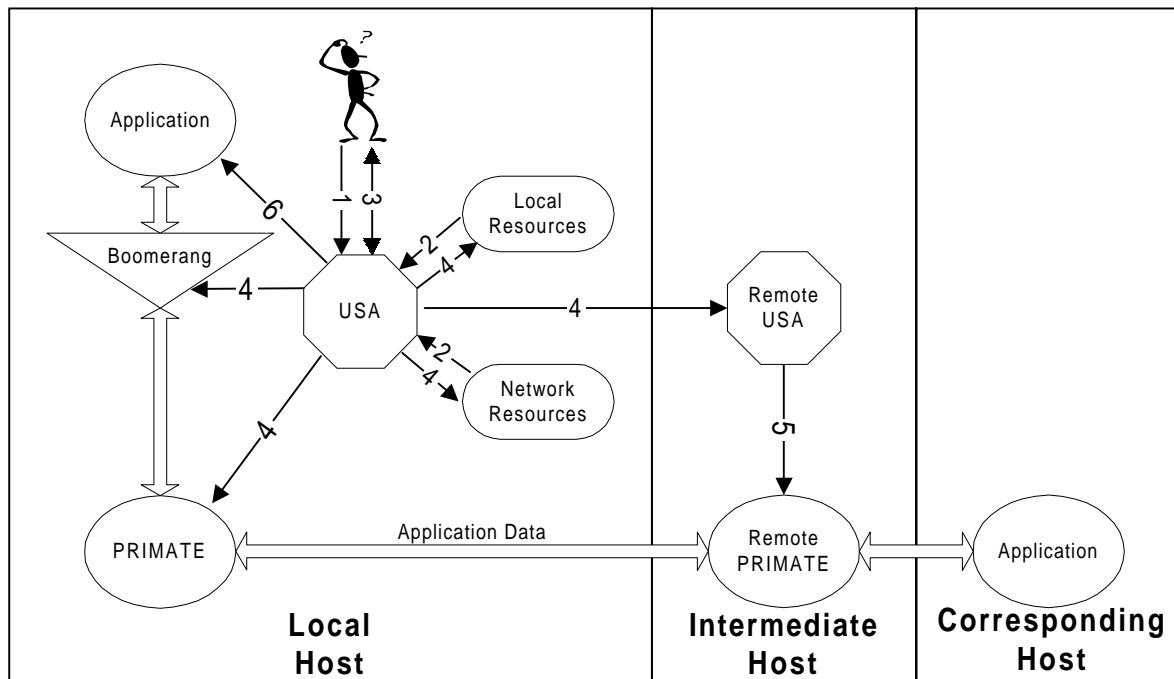


Figure 5 TOMTEN Application Start Up Sequence

Once the application is launched, USA moves into the background. As USA does not try to make any prediction of the perceived level of quality, it does not attempt to modify the environment without user intervention.

When the user is unsatisfied, s/he indicates dissatisfaction by pressing the QoS button (USA's GUI) (1), as shown in Figure 6. TOMTEN then tries to determine what may have caused user dissatisfaction by determining the availability of system resources (2), and comparing it with the resources that were available when the application was launched. This is done by assuming the resource that has the lowest availability gives rise to unsatisfactory performance.

USA then requests PRIMATE to suggest different application adaptation options (3) that will reduce the usage of this "bottleneck" resource. Using this information, PRIMATE will compute and return to USA, where possible, a set of recommended actions, as well as an indication of the bottleneck resource. For example, if PRIMATE detects a reduction in bandwidth, it would recommend that a network with higher bandwidth be used². This information would otherwise be difficult for USA to obtain.

The recommended action will provide an adaptation and/or changes to system resource allocations (4). The application adaptation options will depend on what is supported by the PRIMATE. The system resource reallocation options will be derived from the information obtained resource monitoring. Once the user has made a decision, USA informs the appropriate components of TOMTEN of the decision and requests the appropriate changes (5 & 6).

We emphasize firstly, that there is no resource re-negotiation, as it is generally very complex. TOMTEN therefore uses application adaptation and resource changes that *do not require re-negotiation*. Secondly, the final decision as to what action is taken is made by the user. Thus in the TOMTEN reactive framework, the user can ignore inappropriate suggestions, while in a predictive system, the user unable to prevent inappropriate changes from taking place.

² Assuming that access to a higher bandwidth network is available

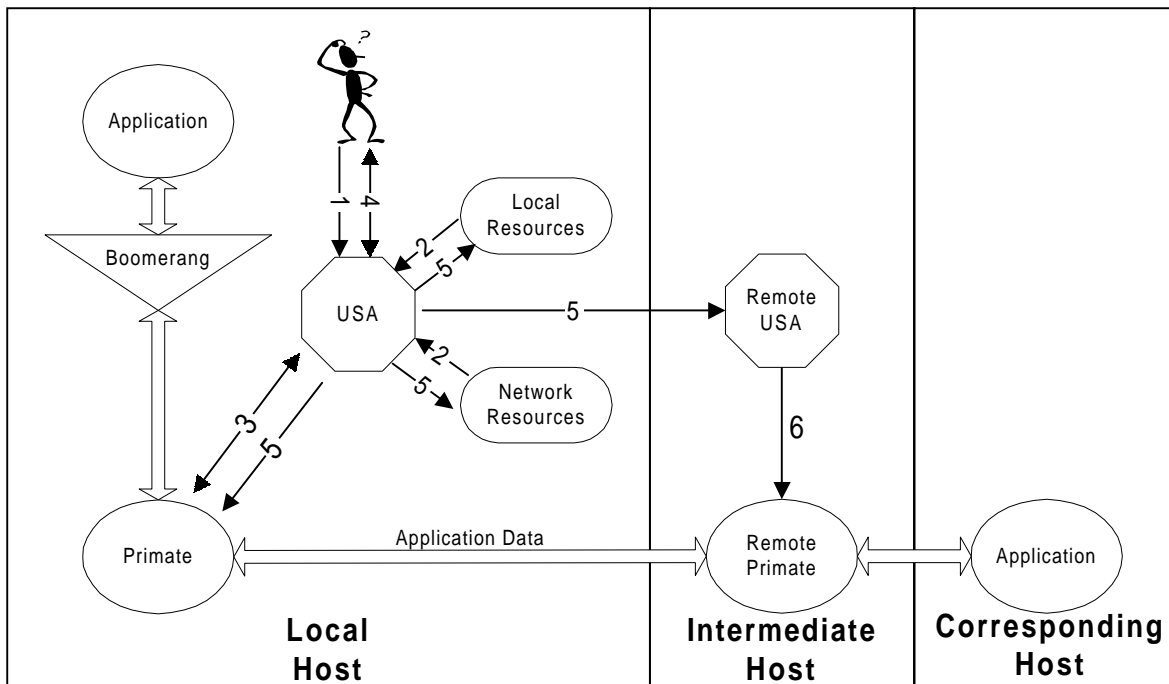


Figure 6 TOMTEN User Dissatisfaction Sequence

4. Prototype Implementation and Experimental Results

As TOMTEN, allows all final decisions to be made by the user, it eases the precision required in the suggestions it makes. Therefore, the development of a working prototype implementation of TOMTEN is considerably simpler than that of other QoS schemes that have been reported. This section discusses the development and testing of a prototype implementation of TOMTEN. We will firstly discuss how the different components of TOMTEN were implemented before explaining experiments conducted with a MPEG and web browser application.

4.1 Prototype Implementation of TOMTEN

The prototype implementation of TOMTEN was developed on the Linux platform. We have currently implemented prototypes of the USA, PRIMATE and Boomerang components of TOMTEN. The implementation architecture of TOMTEN is shown in Figure 4.

The network setup used for the experiments is shown in Figure 7. It consists of a mobile computer that is connected to a base station through three different interfaces – a 10 Mbps Ethernet, a 2 Mbps Wavelan and a 9600 bps GSM link.³

USA was implemented in JAVA apart from the network probes, which were implemented in C for performance reasons. The network probes are currently very simple and allow us to determine which is the most suitable link. They operate by transferring a block of data from the local host to the intermediate host. They currently record the available bandwidth, the round-trip time and packet loss for these short transfers. Different size transfers are done for each of the networks to take into consideration the maximum bandwidth and all transfers are bounded by a maximum delay. It is our

³ The GSM link is simulated using a fixed serial link between the mobile host and the base station.

intention within the TOMTEN framework that the probing only takes place when the user indicates dissatisfaction in order to save on over-head traffic. Hence, it is important that the monitoring process is completed within a reasonable time to obtain acceptable response times⁴.

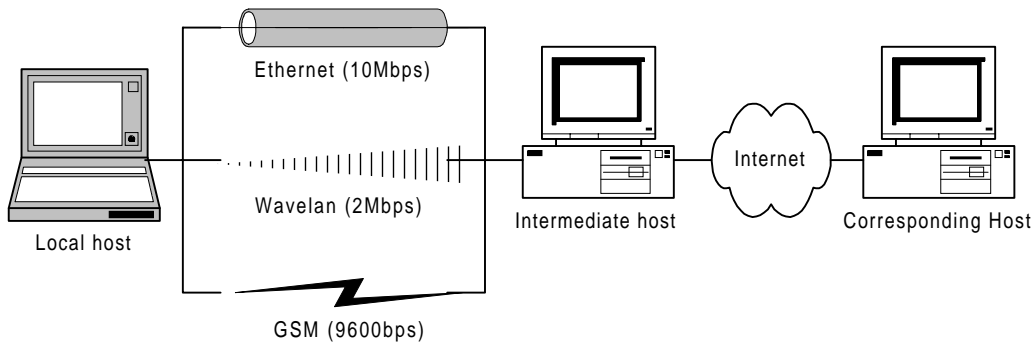


Figure 7 TOMTEN Implementation Testbed

The PRIMATEs have been implemented in C although we are planning to port them to JAVA to introduce platform independence and code portability. The PRIMATEs use sockets to communicate with USA. We discuss the implementation of the PRIMATEs in detail in the following sections.

Boomerang has been implemented as a module within the Linux kernel at the socket layer. It detects socket calls made by the application, and redirects them to a specified PRIMATE port. USA and PRIMATEs use a library of system function calls to communicate with Boomerang. We also believe that it is possible to develop Boomerang as a dynamic link library that will be called instead of the system socket libraries. This will make Boomerang easier to port between different platforms.

4.2 MPEG Player Application

The primary aim of this prototype implementation of TOMTEN was to verify the feasibility of the framework and to evaluate the overhead it introduces. The application we used to evaluate the overhead introduced by the framework was a MPEG player. This MPEG player is based on the MPEG-1 standard that only includes video sequences. It was the attribute of the MPEG standard, to define frames of different levels of detail (I, P and B frames), that allowed us to customise the MPEG player to provide varying levels of quality to the user [12].

To support adaptivity for this MPEG player, we used an automatically generated tailored protocol between the local and remote PRIMATEs. The tailored protocol allows the use of multiple data channels, each supporting their own protocol. In addition, it manages the data channels using out-of-band signalling on a control channel. Further details of the synthesis and operation of the protocol can be found in [4]

The MPEG player uses three data channels to transfer the different frame types as shown in Figure 8. The channel containing the I-frames always uses a reliable protocol (i.e. ordered and error free) as these frames are compulsory for the MPEG application to correctly interpret the data. By defining the protocols used on the remaining P and B channels, we are able to determine the level of detail viewed by the user.

Currently our MPEG PRIMATE has three levels of service – a Full service, a Partial service and a Filtered service. The Full service defines all three channels to be reliable, therefore guaranteeing that all frames will be delivered and viewed. The Partial service defines the latter two channels as

⁴ Need to take into account the rate of change of network conditions as well. This is currently being investigated.

being unreliable (i.e. unordered and no error recovery). Using the Partial service, if the network is congested or the application is slow processing the MPEG clip, P and B frames will be lost, reducing the level of detail presented to the user. This level of service provides partial ordering and partial reliability as discussed in [5]. The Filtered service defines the latter two channels to be filtered (i.e. data dropped at the source). This provides a low quality video consisting of just I-frames.

Our testbed consisted of three PCs (a 486 and two Pentiums) running Linux 2.0.33 and connected through a dedicated Ethernet network. The MPEG Player was based on the Berkeley Player [2] extended into a client-server application. As shown in Figure 8, the local PRIMATE and the MPEG Client coexisted on the client machine. The MPEG Server was on a remote server while the remote PRIMATE ran on an intermediate system. The remote PRIMATE could also have run on the remote server.

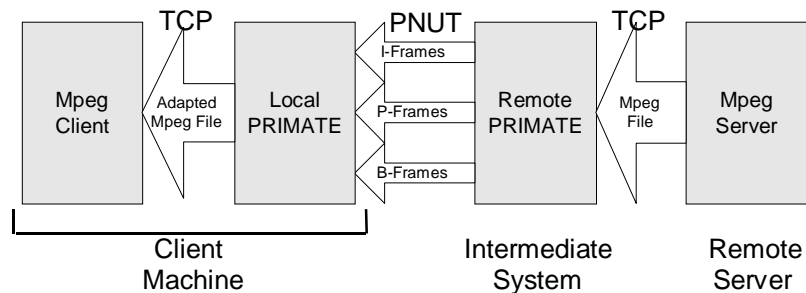


Figure 8 MPEG Application made Adaptive

The following measurements were taken for an MPEG file (consisting of 161 I-frames, 160 P-frames and 639 B-frames) firstly using a direct connection from the MPEG client to the server without using TOMTEN and secondly using TOMTEN with MPEG PRIMATES. The results of these measurements are shown in the table below.

	Frame rate (fps)	Play out Time (seconds)	Frames Processed Total
Direct Connection			
full video (I,P,B)	12.4	77.23	960/960
TOMTEN			
Full video (I + P + B)	12.3	77.75	960/960
Partial video (I + B? + P?)	12.4	67.25	832/960
Filtered video (I)	3.6	45.63	162/960

By comparing the times of the full video play-out when using the direct connection and with the use of TOMTEN, we were able to show that there is only a small increase (0.6%) to the play out time. This additional delay is caused by redirecting the traffic through the two PRIMATES. The CPU usage was dominated by the MPEG application in both cases (85%) as opposed to the PRIMATE only introducing a 2% CPU overhead. Furthermore the differences between the play-out times and the number of frames processed in the three different service levels illustrate that TOMTEN was able to modify the content of the MPEG stream and hence the perceived quality to the user.

4.3 Web browser Application

One of the most commonly used Internet applications, today, is a web browser. We therefore decided to develop PRIMATES to allow application adaptation to be introduced into web browsers. The Web PRIMATE processes the images that the browser receives, similar to the work conducted by [8]. The processing of the images are conducted on the intermediate system as shown in Figure 9. Currently the Web PRIMATE modifies JPEG and GIF images and is capable of reducing the number of colours in the image or by converting it into greyscale. The Web PRIMATE currently does not attempt to resize the pictures as done in [8].

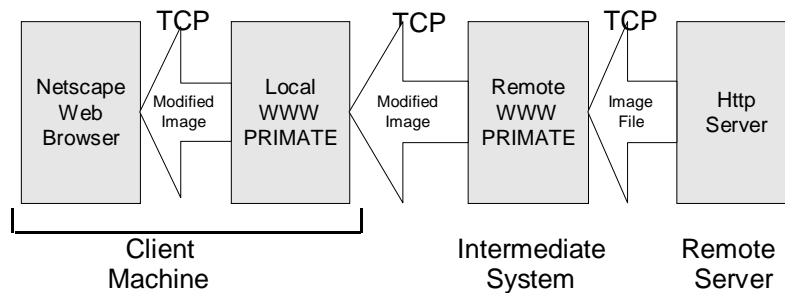


Figure 9 Web Browser Application made Adaptive

When the web browser requests a file, the socket call is intercepted by Boomerang and transferred to the local Web PRIMATE. The Web PRIMATE then relays the request to the Remote Web Primate. With the Web PRIMATES, the main processing is done at the Remote PRIMATE and the local PRIMATE mainly serves as a relay. The Remote PRIMATE requests the HTML file from the HTTP server and checks the stream header to detect if it contains an image. If the image is not found or the stream is unrecognised, it is allowed to pass without modifications. If an image is detected and the PRIMATE has been requested to reduce the incoming image size then the file is saved locally. Currently the remote PRIMATE does not process the image stream, rather it downloads the entire file and uses image processing tools to reduce the size of the image. The modified image is then sent to the local PRIMATE who relays it to the Web browser to display. As it can be seen from this example, it is not necessary to modify the Web browser or the HTTP server. Furthermore, the Web PRIMATE can be used with any web browser.

As the Web PRIMATE, receives the entire image file, processes it and then forwards it on to the Web browser, there is an overhead associated with the Web PRIMATE. This overhead is related to the cost of downloading the file, which is dependent on the size of the file and speed of the link between the Remote PRIMATE and the HTTP server. Then there is an additional overhead for processing the image. These overheads are countered depending on the speed of the link from the intermediate host to the local host and the amount the image size is reduced. Therefore, the benefit of using this Web PRIMATE occurs when there is a slow link between the local and intermediate hosts and the faster connections from the intermediate host to the remote host.

We conducted experiments by running a web browser using the GSM link (9600 bps) of our experimental test-bed to measure the benefits of using the Web PRIMATE. We used the Netscape Web Browser 4.04 for our experiments. Firstly, we measured the time it took from when the request was received from the web browser until it was completed. This time included the transfer from the HTTP server, the processing of the image file and the transfer from the Remote PRIMATE. We also separately measured the times on the Remote PRIMATE to record how long it took to receive the file from the HTTP server and the time to process it.

Two full colour images were transferred – a GIF and a JPEG. Colour reduction processing and grey-scale processing were done for both images. During the experiments, we tried to limit the

image processing so that the resultant image was still clearly recognisable as a reduced quality version of the original. The results are shown in the table below.

	Image Size	Total Time	Processing Time
GIF Image http://silicon.ee.uts.edu.au:81/images/opera1.gif			
Normal	60,624 bytes	78.55 secs	None
Reduced Colours	19,377 bytes (67% reduction)	23.34 secs (70% reduction)	2.058 secs
Greyscale	16,059 bytes (73% reduction)	19.78 secs (74% reduction)	1.739 secs
JPEG Image http://silicon.ee.uts.edu.au:81/images/opera.jpg			
Normal	17428 bytes	19.50 secs	None
Reduced Colours	8036 bytes (53% reduction)	10.26 secs (47% reduction)	0.338 secs
Greyscale	6912 bytes (60% reduction)	9.15 secs (53% reduction)	0.267 secs

The results show that great improvements are possible by using the Web primate on a slow link. These benefits will be reduced when using higher speed links since the size reduction techniques currently used were very simple. Further reductions could easily be achieved by scaling down the large pictures as shown in [8].

5. Conclusions

In this paper, we have introduced TOMTEN, a framework for managing resources in order to provide maximum user satisfaction. The essential characteristics of TOMTEN are:

- It provides a complete framework that supports application adaptivity and application-specific re-configurable protocols. We do not know of any other framework that attempts to address all these issues.
- TOMTEN de-couples QoS management and application adaptation by separating the functionality into USA and PRIMATE respectively. This decreases the complexity of adaptive designs and provides structured methodology for incorporating adaptivity.
- It proposes the use of a reactive QoS management paradigm, which is unique, and facilitates easy implementation in contrast to the schemes that have been proposed to date. This paradigm, USA differs from other QoS management models by not attempting to predict users' perception of quality.
- Application adaptation is made transparent to the application through BOOMERANG, thus enabling the use of applications with no modification.

We have shown through the development of the prototype implementation of TOMTEN, that it is feasible and easy to realise. We have also shown through the development of two PRIMATE

modules, one for MPEG Player and one for Web browser, that we are able to transparently provide application adaptation.

Finally, the only research which has similarities with TOMTEN that we currently aware of is the work proposed by [11]. However, their QoS management scheme, like all previous proposals, is predictive as opposed to USA that is reactive. Furthermore, their work does not support configurable protocols, the use of intermediate systems or downloadable proxies. Therefore, we believe TOMTEN is the first framework, which uses a reactive paradigm end system resource management.

Acknowledgements

We would like to acknowledge the financial support from Ericsson Australia and the Australian Research Council.

References

- [1] A. Bakre and B. Bradrinath, I-TCP: Indirect TCP for Mobile Hosts, in Proceedings of 15th Intl. Conf. on Distributed Computing Systems, May 1995.
- [2] Berkeley Player, Berkeley Multimedia Research Center (BMRC), University of California, Berkeley, 1996. http://bmrc.berkeley.edu/projects/mpeg/mpeg_play.html
- [3] H. Cho and A. Seneviratne, User Centric QoS Management Framework and Its Implementation, UTS Research Report UTS-EE-97-58. Sydney (Australia). November 1997
- [4] R. De Silva, PNUT - Protocols configured for Network and User Transmissions, PhD Thesis, University of Technology, Sydney, Jan. 1998, submitted for assessment.
- [5] M. Diaz, C. Chassot and A. Lozes, From the Partial Order Connection Concept to Partial Order Multimedia Connections, in Proceedings of HIPPARCH Workshop, Dec. 1994.
- [6] C. Diot, Adaptive Applications and QoS Guarantees, IEEE MMNet, Aizu, Japan, September 1995.
- [7] K. Fall, J. Pasquale and S. McCanne, Workstation Video Playback Performance with Competitive Process Load, Fifth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '95), Durham, New Hampshire, USA, April 1995.
- [8] A. Fox, S. D. Gribble, E. A. Brewer and E. Amir, Adapting to Network and Client Variability via On-demand Dynamic Distillation, in Proceedings of 7th Intl. Conf. On Arch. Support of Prog. Lang. And Oper. Sys. (ASPLOS VII), Oct. 1996, Cambridge, MA.
- [9] R. H. Katz and E. A. Brewer, The Case for Wireless Overlay Networks, SPIE Multimedia and Networking Conference (MMNC'96), San Jose, CA, USA, 1996.
- [10] B. Landfeldt, A. Seneviratne and C. Diot, User Services Assistant: an end-to-end reactive QoS architecture, IFIP 6th International Workshop on Quality of Service (IWQoS'98).

- [11] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn and K. R. Walker, Agile Application-Aware Adaptation for Mobility, to appear in the Proceedings of the 16th ACM Symposium on Operating System Principles, 1998.
- [12] K. R. Rao and J. J. Hwang, Techniques & Standards for Image Video & Audio Coding, Prentice Hall, 1996.
- [13] A. Vogel, et al., Distributed Multimedia and QoS: A Survey, IEEE Multimedia, Vol. 2, and No. 1, summer 1995.