

# MARCH: A Distributed Content Adaptation Architecture

S. Ardon<sup>1,2</sup>, P. Gunningberg<sup>4</sup>, B. Landfeldt<sup>3</sup>, Y. Ismailov<sup>3</sup>,  
M. Portmann<sup>1,5</sup>, A. Seneviratne<sup>1</sup>

<sup>1</sup>School of Electrical Engineering and Telecommunications  
University of New South Wales, Sydney NSW 2051 - Australia  
{a.seneviratne, ardon, m.portmann}@unsw.edu.au}

<sup>2</sup>LIP6 - Université P. et M. Curie  
Equipe Réseaux et Performances  
8, rue du capitaine Scott 75015 Paris, France

<sup>3</sup>Ericsson research Networks and Systems  
Torshamnsgatan 23 - 164 80 KISTA Sweden  
{bjorn.landfeldt, yuri.ismailov}@era.ericsson.se

<sup>4</sup>Department of Computer Systems  
Uppsala University  
Box 325, S-751 05 Uppsala, Sweden  
perg@docs.uu.se

<sup>5</sup>Computer and Networks Laboratory TIK  
Swiss Federal Institute of Technology ETH  
CH-8092 Zürich, Switzerland

**Keywords:** proxy architectures, content adaptation, overlay network

## 1 Introduction

Over the past years, the number of hosts connected to the Internet has grown tremendously. With the introduction of the 3rd generation and next mobile systems, this number is expected to grow further. At the same time, terminals connected to the Internet have and will continue to have very different capabilities in terms of display capabilities, CPU power, memory, and available power. The constant demand for new distributed multimedia applications and the desire to use these applications with any type of terminal, using any type of access network, introduce new challenges in the delivery and presentation of content to the users. Thus, it will be necessary to develop mechanisms for adapting the content to match the requirements of each permutation and combination of application, terminal and network type.

The Mobile Aware Server Architecture (MARCH) is a distributed Content Adaptation architecture which adapts multimedia content in client-server environments, and the transmission of this content, to optimally match the access network capabilities of various terminal devices, and the user preferences [1,2]. MARCH enables the creation of a service-based overlay network, by the use of intermediary entities, namely proxies, which are deployed at various points along the data path between the client and the server, like most other content adaptation frameworks that have been reported in the literature. However, in contrast to these, MARCH adopts a server-centric approach,

which provides a number of significant advantages. Firstly, if we consider the copyright issues surrounding content transformation in the Internet, it is obvious that a server-centric decision is superior since the transformation is done under the control of an authorized provider of this content. Furthermore, as the server has a better global knowledge of the overall system state, an intimate knowledge about the application being invoked.

The knowledge at the server is exploited by the MARCH in two ways:

1. The use of a *Dynamic Proxy Execution Environment* for dynamic use of proxies. The Dynamic Proxy Execution Environment is an environment which facilitates the uploading of proxies on a per-session basis and using them for multiple sessions, thereby allowing the required proxies to be placed at optimal points along the data path.
2. *The use of a Front-end to the Content Servers* for providing seamless integration. These front end servers, referred to as the Mobile Aware Servers (MAS), make the decision as to which proxies to use for a given set of conditions, and where to execute these proxies. Thus the MAS coordinate the deployment and activation of the proxies to be used in a particular session in a centralized manner.

This paper describes the operation of the MARCH framework in details. Specifically, how the MAS operates and how it is used to dynamically deploy proxy services. Then, through a prototype implementation, it demonstrates the ability of the server-centric architecture to exploit the knowledge at the server to provide optimal adaptations, and demonstrate the use of a suite of content adaptation functionalities for popular client-server applications. In addition, it illustrates the scalability of the framework especially in terms of the signaling used between various components of the framework.

The remaining of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the March framework details; Section 4 introduces our prototype implementation and presents the experimental results. Finally, section 5 presents our conclusions and directions for future work.

## **2 Related Work**

Problems associated with the heterogeneity of the Internet have lead to various proposals in the areas of Active Networks, Active Services and Application Layer Active Networking. These proposals, similarly to MARCH, address the heterogeneity problem by adapting the content and its transmission to suit the operating conditions. In this section, we review the existing approaches in this area and highlight the differences between the server-centric approach of MARCH and these proposals.

### **Content Adaptation with Active Network or Active Services**

Content adaptation schemes that have been proposed which operate at the network level are often referred to as "Active Network" proposals. The concept of Active Network was originally proposed in [3]. It is defined as a network in which the "routers or switches of the network perform customized computations on the messages flowing through them". This concept can be thought of at one extreme as representing the capsule or the programmable switch model where, packet headers carry code to be executed on the routers or switches. The code specifies how the packet should be handled. In the programmable switch model, switches can be extended to perform computations on packets, but the code is not necessarily carried in each packet. Proposals such as [4,5] use this approach to provide content adaptation functionality. These schemes suffer from three major

drawbacks. Firstly these schemes do not scale well. Secondly, introducing high-level functions such as content adaptation at the network layer compromises security and robustness. Finally and most importantly, the models require changes to the network elements and therefore prevent an incremental deployment.

Active services [6] or Application-Layer active networks [7] deploy high-level services at the application layer. They are built on top of the existing robust and simple Internet routing subsystem. Thus, the network layer remains unchanged, facilitating incremental deployment on top of the current Internet architecture [6]. In both these proposals, the adaptation mechanisms to deploy are still embedded in the network in-between the client and the server. They simply operate at a higher layer at the nodes within the network. Thus they are an extension of the active network model in that the decisions are done by network elements. Although they overcome some of the problems associated with classical active networks, the many-to-many relationship that the network elements need to maintain to adapt any type of content to any network characteristic, terminal type, and user preferences, makes the decision at best difficult and inefficient, and more likely intractable.

There is also work in progress within the Internet Engineering Task Force (IETF) aimed at defining an open platform to execute services at arbitrary intermediaries in-between the client and the server such as Web Intermediaries working group (WEBI) [8], Internet Content Adaptation Protocol (ICAP) [9], and more recently the Open Pluggable Edge Service (OPES) working group [41]. WEBI is concerned more with content caching rather than adaptation, ICAP is considered as one possible signaling protocol that can be used for adapting HTTP traffic only. Therefore, they do not provide a general purpose content adaptation framework which can be used with any distributed multimedia application. OPES is still in its infancy, no particular protocols have been defined, and no particular architectural choice of where to place the decision process have been formulated.

Finally, adapting content to client and network characteristics at the server itself has been studied in several proposals such as [10] and [11]. In some of these proposals, the content is authored in a generic, high-level format such as the eXtensible Mark-up Language (XML), and then rendered through style sheets to cater for a particular format making it presentable on a specific terminal. Although these methods provide a framework that can be used when creating new content, they cannot be used with content that have already been created. Moreover, these solutions are restricted to content adaptation, and cannot provide network service enhancements.

### **Content Adaptation with Static Proxy services**

Proxies are application-layer software entities that request or process content on behalf of the clients. Some proposals such as [11,12,13,14,15,16,17,18] and even commercial products [19,20] explore the use of proxies to adapt multimedia content to client and network variations.

These systems which use proxies to deal with client and network variation can be classified in three groups:

*Protocol Enhancement Proxies:* These proxies provide mechanisms that can be used to enhance the performance of a given protocol operating in a particular network. A typical example is the Berkeley snoop scheme [21] which provides mechanisms that improve performance of TCP when operating in lossy networking environments (e.g. wireless). Recent work in the IETF in the Performance Implications of Link Characteristics (pile) working group summarizes the advances in this area. A more complete survey of available techniques can be found in [22]. Proxies can also be used to implement tailored communications protocols to cater for particular link characteristics. Examples of such systems are presented in [23].

*Media Transcoding Proxies:* These transform one media type to another, e.g. HTML to WML. This type of transformation is often used when the terminal characteristics prevent the content to be presented in its original format.

*Content Distillation Proxies:* These reduce the bandwidth requirement of data objects by applying transformations that utilize the knowledge of the content format and possibly its semantics. An example of content distillation is bit rate reduction in streaming video by dropping entire frames or reducing the resolution. Distillation is often useful in situations involving low bit rate networks, but also to cater for different terminal characteristics [12]. A number of proposals have focused on providing content distillation systems for heterogeneous network and clients, specifically for HTTP traffic [12,15,16,17,18].

In these fixed or 'static' proxy architectures, the client application requests are directed to the proxy either explicitly (e.g. the user manually configuring the proxy settings in the client application) or transparently as described in [24]. Each of these solutions is designed to perform a specific type of adaptation, to cater for a specific set of requirements. Combining these proxy services together to provide a composite service is impossible because of the static nature of the implementations.

### 3 The MARCH Framework

#### 3.1 Architecture overview

The type of adaptation to be performed for a given session<sup>1</sup> depends on the *operating conditions* of the session. We define the *operating conditions* as the combination of characteristics of the access network the client is using, the characteristics (CPU/Memory/Screen Size) of the client terminal device, and the user's preferences.

One of the key design goals of MARCH is to allow the composition of a service to provide the optimal service for a given set of *operating conditions*. Services are composed by concatenating a number of proxies to form a global, multifunctional service. An example of such a service is a "content distillation and encryption" service. The architecture needs to maintain the integrity of the system if multiple proxy functions are carried out successively. For example, in the case of the "content distillation and encryption" service, the content distillation needs to take place before the data is encrypted. Once the data is encrypted, functions requiring knowledge of the data semantics cannot be used as these semantics are hidden. In the March framework, the decision as to which proxy modules to use and in which order to use them is made centrally, at the content server. This allows the framework to maintain the integrity of the system.

The architecture of the March framework is composed of three main administrative domains, as shown on Figure 1: the client, the server and the service provider administrative domains (for simplicity, Figure 1 shows only one service provider, but several can be used successively in the application data path). The collection of service administrative domains constitutes the service overlay network. The server administrative domains can invoke content adaptation service instances within these service administrative domains.

---

<sup>1</sup> The notion of session is dependent on the application used. For example, in the case of an audio-on-demand application, the session is defined by the duration of the transmission and presentation of the audio file. In the case of a web browsing application, the session could be defined by the duration a user is using a particular website services

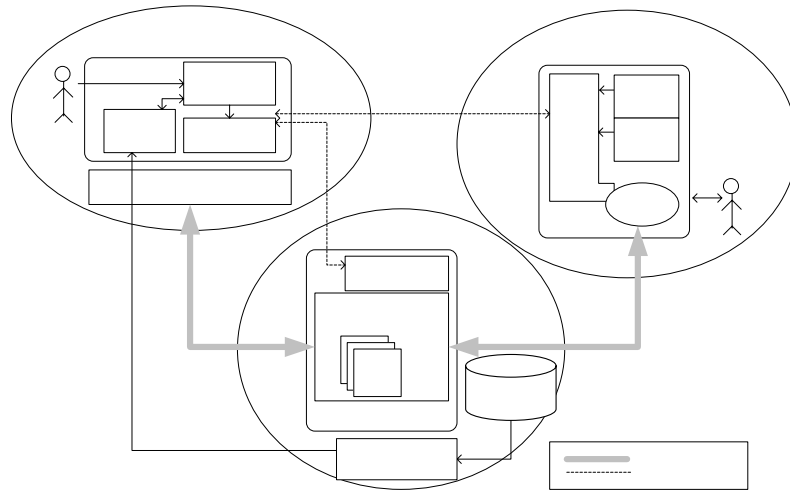


Figure 1: MARCH framework architecture

The architecture is composed of four main components distributed across the network. In the following sections, we describe these components and their functional interrelationship.

### 3.2 Compute Servers (CS)

The CSs provide the dynamic proxy execution environments. The proxies to be used can either be dynamically inserted into the CSs execution environment on a per-session basis, or housed at the CSs on permanent or semi-permanent basis. These proxies will then be concatenated to compose the required service.

The CSs can be located anywhere in the network as well as at the client and the server. In the network, there are several natural points where CS would be located, such as network boundaries / performance discontinuities, and content provider premises. It is possible for two types of concatenations to take place, namely a number of proxies within one CS and number of proxies in more than one CS. We refer to the concatenation of proxies within one CS as a *proxy chain* and concatenation of proxies in more than one CS as a *proxy path*.

### 3.3 The Mobile Aware Server (MAS)

As described earlier, the MAS is a front end to the content servers. Thus it is under the same administrative authority as the content server. The MAS is responsible for determining what adaptation functions are required to provide the optimum service for a set of *operating conditions*, and coordinate the deployment of these adaptation functionalities. In other words, it needs to decide what proxies to use as well as choose the Compute Servers on which the proxies are to be executed. This is done by a *decision engine* within the MAS. MAS uses two types of inputs to make this decision. Firstly, it uses the information obtained from the client when a session is requested, namely the terminal capabilities, access network characteristics, and user preferences. Secondly, it uses specific knowledge available at the server about the application, specifically media type and server architecture (e.g. replication of content on different servers).

Cont

MAS

Service  
Discovery

Cont

## Decision Engine

The MAS decision process can be automated. However, we believe that the development of an automatic decision making process is complex and unnecessary. The same functionality can be realized through a manual configuration of a rule-based system. This manual configuration will be done by the server administrator, similarly to a network administrator setting up an access control lists. The rationale is that only the content provider can decide which adaptations are acceptable under different *operating conditions*. With the manual configuration, the server administrator can specify the proxy configuration for all *operating conditions* that the server wishes to support. More importantly, fully automatic decision process and service composition require the proxy services input/output to be formally characterized [40]. Some proposals attempt to use simple data types such as MIME types for this purpose, but this fail to characterize even simple content distillation services, as the quality factors cannot be extracted from these simple types. Manually defining the sequence of adaptation services to apply to application data allows the administrator to control the service composition, using any type of service [40].

Compared to automatic or semi automatic decision schemes as described in [25], it can be argued that the manual configuration limits the number of *operating conditions* that can be supported, and the granularity of the adaptation services offered. We contend that this is not the case, as it is possible to aggregate and generalize the *operating conditions* to provide finer adaptation granularities where necessary. For example, the rule “all screen size inferior to HxW pixels, use adaptation proxy A” allows several operating conditions to benefit from one service proxy. The rules are authored using the emerging RuleML rule description language [42], and we propose to use high-level rule authoring tools to reduce the risks of errors [40]. To obtain information on which proxy modules are globally available, the administrator will be able to use a service discovery system, similar to the yellow pages service.

The decision engine obtains the information about the terminal capabilities and user preferences from clients, when a connection is requested by a client. The information is communicated between the client and the MAS using the Resource Description Format (RDF) [26] as described in [27]. The access network can generally be characterized by the link bandwidth, delay, and loss. These can be determined through a monitoring process [29,28]. The decision as to where to place the proxies depends on the precedence relationships of the proxies and the load on the available CSs. The determination of the CS load can be incorporated in the monitoring scheme as suggested in [30].

The rule set is evaluated sequentially at session setup time with the information communicated by the client and the monitoring process. The result of this evaluation is the ordered set of adaptation proxies and compute servers to use the session [40].

### 3.4 The MARCH Client Entity (MCE)

The MCE conveys information about the terminal capabilities to the server though an out-of-band signaling mechanism. To facilitate this, the MCE is located at or close<sup>2</sup> to the terminal and uses a description of terminal characteristics authored by the user/administrator. The Composite Capabilities/Preference Profile (CC/PP) [39] is used to specify the format of this description. In addition to that, the MCE directs the upstream application traffic to the correct proxy, where necessary. The interception of the application requests and its possible modification is application-specific. This is therefore done using an application-specific plug-in within the MCE. An example

---

<sup>2</sup> E.g. in a bearer service. For example a corresponding WAP phone would not be able to run a MCE, but this MCE could be located in the WAP gateway



As explained earlier, the MAS performs the decision as to which service provider to use for the particular session, using the decision engine. The MAS then request the instantiation of these proxies by either installing new proxy-modules on one or more CSs, or by using a set of previously instantiated proxies and configuring them to perform the proxy concatenation. In the first case, the MAS requests the CS to instantiate the chosen proxy-modules (4). The CS in turn fetches the proxy modules (5, 6), perform admission control, and if successful, instantiates the proxies. Furthermore, it informs the MAS of the operation' success (7). The operation is then similar to the case when using the pre-installed proxies. In both cases, the MAS return information about how to connect to the first-hop proxy, to the MCE (8). Finally, the client application can begin its operation as its requests are directed to the first-hop proxy through the Application Specific Helper (ASH).

The interactions between the different components of the architecture require the use of signaling protocols, which we describe in the next section.

### 3.7 Signaling in March

From the above description of the framework operation, it can be seen that MARCH not only requires signaling between the MCE and the MAS, but also between the MAS and the CSs. This is illustrated in Figure 3.

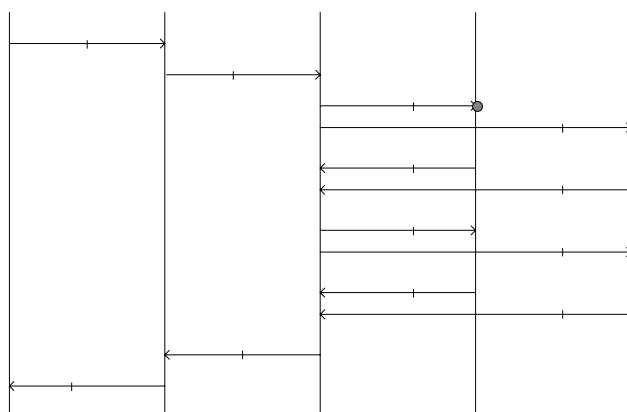


Figure 3: Signaling in March

The MCE-MAS signaling, as described earlier, enables the MCE to inform the MAS of the user preferences, terminal features and the access network type. The MAS-CS signaling allows the installation and concatenation of proxy modules into a *proxy path*.

#### MCE to MAS Signaling

As described in the previous section, the MCE to MAS signaling is facilitated by the MCE transparently intercepting all client application's request to the content server. After the MCE intercepts a request (1), it generates a signaling message to the MAS, providing it with information available at the client, namely the terminal equipment and access network characteristics, and the user preferences (2). If there is no MAS at the server, the MCE simply forwards the original application request directly to the content server and no MARCH functionality is activated.

## MAS - CS Signaling

MAS to CS signaling facilitates the installation and configuration of the proxies to be used for a session. This is done by the MAS acting as a central controller of the concatenation process. The signaling associated with creating *proxy chains* and *proxy paths* is illustrated by messages 3 to 6 in Figure 3

First, the MAS signals all the involved CSs which proxies should be used (3). At point A, the CSs download the required proxies (not shown on the figure), configure them and allocate the necessary network service points (TCP and/or UDP sockets) for the proxy chain. At this point, admission control can be performed to accept or reject the new service requests. The network service points are defined using the 3-tuple IP address, port number, and protocol type (UDP or TCP), uniquely identifying where the proxy chain is listening for connections and/or receiving data.

To instantiate a proxy path, the service points for a given proxy chain are communicated to the neighboring proxy chain(s) in the proxy path. As the port numbers for server sockets are dynamically allocated by the CS they need to be sent back to the MAS by the CSs (4).

Second, once the MAS has collected all the necessary information about the network service points, it individually signals each CS with the network service point details of the neighboring CSs in the proxy path (5). This configuration request is then acknowledged by each CS (6). Once the MAS has received acknowledgements from all CSs, the proxy path is completely configured. The MAS then sends back a Client Reply message (7) to the MCE, and conveys where to connect, i.e. the network service point details of the first proxy in the proxy path.

### Discussion

In section 2, we listed some of the problems associated with using static proxy solutions. As described above, MARCH overcomes these problems by using a server centric design, and dynamic system architecture.

Firstly in MARCH, there is no intermediate proxy at a fixed Location. The MAS location is the same as the content server to which the connection is to be made. Therefore, there is no need for the users to obtain any specific information from the network administrator to configure the client terminal. Moreover, users neither have to manually configure their terminal devices, nor configure any application. Finally, once a session has been established, and torn down, the configuration for the next session is done automatically at session establishment time, i.e. there is no need to reconfigure.

Secondly, MARCH provides the infrastructure to concatenate general-purpose proxy modules in order to compose the optimum service for a given set of operating conditions. Thus, new functionality can be provided by adding new modules. Moreover the open MARCH architecture enables third party vendors to develop specific modules to provide customer-specific services. This allows for example, terminal device manufacturers to develop proxy modules to perform specific tasks optimized for new terminals as they are introduced to the market.

Thirdly, MARCH provides better security management features. The MAS is able to ensure the precedence constraints of the proxy modules as mentioned in section 3.3 as it controls the placement and the concatenation process. For example, the problem of end-to-end security and proxies can be solved in MARCH since the MAS is able to place the proxy functionality at the server and the client. MARCH can also provide better control over security, as the MAS can house the proxies in secure or trusted CSs when necessary, as described in [31].

Fourthly, the concatenation of proxies in MARCH, allows the end users to benefit from multiple functions simultaneously. For example, it is possible to use consecutively a filter that strips advertisements of web pages, a filter that reduces the size of images and transcodes them from colour to greyscale, and a filter providing end-to-end encryption.

Finally, we believe that the unique server-centric approach of MARCH provides numerous other benefits:

- If we consider the copyright issues surrounding content transformation in the Internet, it is obvious that a server-oriented decision is superior since the transformation is done under the control of an authorized distributor for this content [32].
- Considering privacy and security issues, the server is also the most logical entity to control the manipulations to be done on the content.
- When dealing with proprietary media data formats, the content server might be the only entity able to operate on the data being transmitted, it is therefore the most suitable entity to control content transformation.
- The MAS has access to all information about the media before configuring the session. Without this information it will not be possible to correctly adapt the session. For example it is impossible for a client to determine the actual coded bit rate of an mp3 file, by only looking at the mime type or file extension.

## **4 Prototype Implementation and Experimental Results**

### **4.1 Prototype Implementation**

We developed a prototype implementation, to evaluate the feasibility, performance and scalability of the framework. The prototype was built using Java, despite some of its performance implications, because of the resulting simplicity and code portability. We used Java's specific features to dynamically and remotely load code as the foundation for code mobility. The application requests were transparently intercepted at the client using transparent capture utility, referred to as boomerang, that we developed [33]. Boomerang intercepts all network-related system calls from an application and provides stubs to hand over the connections to the MCE. The CSs use java network class loading to dynamically load proxy classes from the network. The Proxy Repository is implemented using the Apache web server, with proxy bytecode bundled in compressed *jar* files.

#### **Service discovery**

Service discovery is implemented using IP multicast. The CSs periodically advertise their address to a multicast group address. We piggyback CSs load information in these advertisement messages. The metric for the load is advertised as the number of proxies running on the CS. In addition, the details of CSs can also be manually entered into the MAS configuration file. This can be used especially for handling CSs associated with fixed proxy services (e.g. hardware and/or software transcoders).

#### **Decision process**

The decision process in the prototype is implemented as a simple table lookup, using a multi-dimensional hash table and a static configuration file defined by the MAS administrator. The configuration file is written using XML, and allows the server administrator to create a tree which describes all sets of operating conditions the system wishes to cater for, and the corresponding

proxy path descriptors. The remaining situations that are not described in the configuration tree are matched using wildcards.

As described in section 3.3, the MAS will decide which CS to use using their current load as the metric. In this prototype, the MAS decides on a compute server by using a simple weighted round-robin election process, where the weights are determined using the load on each CS. Although this does not reflect all the details presented in section 3.3, it is sufficient to evaluate the performances of the system.

### **Signaling**

The signaling for the prototype was implemented using text based messages over TCP. When a connection is requested, one TCP connection is established between the MCE and the MAS for signaling. The MAS in turn establishes another signaling TCP connection between itself and each chosen CS for the session. The delay introduced by these TCP connection establishments can be amortized over several sessions by using pools of pre-established connections.

### **Proxies Modules**

Two categories of proxies were developed as part of the prototype implementation. One group represents utility proxies which have the lowest common denominator structure of many other proxies. These simple proxies are bare forwarding proxies. The simple TCP proxy is a TCP relay, which interconnects two TCP connections. The UDP proxy is similar to the simple TCP proxy, and interconnects two UDP sockets. These types of proxies are used in conjunction with other proxies, both fixed and dynamically loaded, to allow the data streams to be routed correctly through the series of proxies for the creation of *proxy paths*.

The other group represents proxies that provide specific content adaptation functionalities for popular client-server applications. These were implemented to show the feasibility of providing complex functionality with dynamically loadable proxies. To this end we developed the following MARCH proxy modules:

- An MP3 transcoding proxy which transcodes an MP3 audio stream from any given bitrate down to 16kbps. A widely used public domain MP3 encoder/decoder [34] was used to perform the actual transcoding. A Java wrapper was used to pull the stream from the server. A free streaming MP3 server was used as the content server [35] and freeamp [36] was used as the client MP3 player.
- A web cache/transcoding proxy based on the RabbIT transcoding proxy [13]. RabbIT downgrades the quality of images in web pages. Furthermore, it can be configured to remove background pictures, java applets and javascript from HTML pages.
- The TLS proxy terminates a secured TLS connection [37] on one side and splices it to a standard TCP connection on the other side. This allows interconnecting a network in which data needs to be protected to one in which data can be sent unprotected. This enables content adaptation to be carried out by proxies within the trusted part of the network. Thus by concatenating the TLS proxy with any content-adaptation proxy, we can provide content adaptation over secure channels.
- The compression proxy basically provides a “compressed TCP” service. It is implemented using the standard GZIP compression algorithm. The proxy is capable of compressing or decompressing, depending on the arguments it is invoked with.

While using native code to perform the actual transcoding in our proxies diminishes their portability, we observed that the packages successfully compiled on most operating system. Sun's Java Media Framework (JMF), which provides java codecs for various multimedia formats may overcome this problem, but was not used in our prototype implementations for historical reasons. This does not influence the findings as we were evaluating the performance of the architecture as opposed to individual modules.

## 4.2 Performance Evaluation

We conducted a series of experiments to evaluate the performance of the MARCH architecture using the prototype implementation described above.

### Experimental Setup

The experimental setup consists of off-the-shelf PCs for the MAS, CS, Proxy repositories, content servers and client. The PCs are all Pentium III 500 MHz, equipped with 128MB of RAM and interconnected through a 100Mb/s switched Ethernet. They run Linux 2.2.17 and the SUN Java Virtual Machine (JVM) 1.3.0 (just-in-time compiler enabled, native threads). All experiments were conducted on 4 unloaded machines, interconnected via an unloaded LAN.

### Call Setup Delay

The first experiment was to measure the call setup delay of MARCH for various applications and *operating conditions*. The objective was to assess the delays involved with using dynamically loaded proxies. The call setup delay is defined as the delay introduced by MARCH when establishing a session, i.e. the delay between the applications generating a request, (1) on Figure 3 and the application receiving a reply (8), measured at the client. This delay can be analytically represented as (1) on figure 4.

$$2 \cdot rtt_{MCE-MAS} + t_{decision} + \max_i (3 \cdot rtt_{MAS-CS_i} + t_{download_i} + t_{inst_i}) \quad (1)$$

with:

- $rtt_{MCE-MAS}$  : round-trip delay between the MCE and the MAS
- $t_{decision}$  : time taken at the MAS to perform the decision
- $rtt_{MCE-CS_i}$  : round-trip time between the MAS and compute server i
- $t_{download_i}$  : time for the compute server i to download the proxy module(s)
- $t_{inst_i}$  : delay associated with java overheads (e.g. just-in-time compilation) at compute server i

Figure 4: Analytical evaluation of the call setup delay

This call setup delay is one of the primaries overhead associated with architectures such as MARCH, especially in situations where one or more proxies need to be pushed onto the CSs.

For these experiments, the application request resulted in the MAS installing one *proxy chain* consisting of one proxy, in the data path. The call setup delay was measured for one hundred session establishments. Results are presented in Table 1.

	# lines	size (bytes)	with class cache		without class cache	
			$\Delta_1$ (ms)	std. dev.	$\Delta_2$ (ms)	std. dev.
<b>HTTP TLS</b>	517 (1)	430597	75	12	589	35
<b>web transcoder</b>	5393	62 526	51	6	319	14
<b>mp3 transcoder</b>	702	10 527	120	23	259	36
<b>TCP/ZIP</b>	210	3797	49	4	90	15
<b>Simple (TCP)</b>	170	3502	41	5	85	15
<b>Simple (UDP)</b>	127	2177	42	3	84	17
(1): number of lines excludes supporting packages (java security)						

Table 1: Call setup delay for the various proxies developed

The initial session establishment delay,  $\Delta_2$ , includes the delays associated with the CS downloading the proxy code from the proxy repository and the JVM performing just-in-time compilation of the bytecode. The call setup delay in subsequent invocations,  $\Delta_1$ , only represents the loading of the Java classes which are cached at the CS. These results, while dependent on the type of proxy used and their implementation, give an indication on the performance of dynamic proxy installation in MARCH<sup>3</sup>.

First, as expected, in the case when the proxy has to be downloaded, the call setup delay is significantly higher than when the proxy code is cached. This delay is composed of two parts: the time to download the proxy modules and the instantiation overheads associated with Java (including variable overheads of the just in time compiler).

Of these, the primarily contributor is the time take to download the proxy modules from the proxy repository. This can be seen from the correlation between code size and  $\Delta_2$  values in Table 1. In the case of the TLS proxy module this corresponds to more than 25%. The other component consists of some fixed parts and some variable parts. To illustrate this, Table 2 presents a breakdown of the delays for the Simple TCP, TLS and MP3 transcoding proxies.

The fixed parts consist of the signaling and the MAS processing overheads. The variable overheads are mainly due the downloading, namely CS Java network Class loading, Java-related overhead such as the just-in-time compilation and the CS proxy instantiation and configuration. This can be seen in the case of the TLS proxy, where the large amount of bytecode results in a particularly long just-in-time compilation overhead. Similarly, the MP3 transcoder proxy takes significantly longer to initialize itself since it must instantiate an external UNIX process. This is suboptimal, and is expected to be much better if the transcoding is done 100% in java, or using a native library instead. In the case of the TLS proxy, the just-in-time compilation of the java TLS classes contributes to the high configuration overhead. In all three cases, the overhead of the signaling remains significantly lower than the other delays.

---

<sup>3</sup> It is recognized that this will depend on the on the implementation, hardware configuration and other system related factors. To overcome this variability the results are for a single implementation and consistent experimental set-up

As the network was uncongested (the average round trip times was less than 1ms to any of the components), the delay  $\Delta_1$  is primarily due to the implementation overheads associated with Java, with a minor contribution from the signaling.

	Simple TCP	TLS	MP3 transcoder
MCE - MAS signaling	5	4	4
MAS - CSD signaling	8	9	8
MAS processing	13	12	12
CS Java network class loading	38	80	43
CS proxy instantiation,	20	61	17
CS proxy configuration	~0	433	168
<b>Total</b>	84ms	584ms	252ms

Table 2: Breakdown of call setup delay

Thus, the MARCH signaling delays associated with creating *proxy paths* will not have a significant impact on the overall performance of the system as it is negligible especially if the network paths between servers and compute servers are of high capacity. The framework introduces an additional client-server round-trip delay at session setup time, which can be problematic if the access network has a very high latency. However, in this case, this increased delay should be compared with the benefit of the service provided to the user, and the duration of the session.

#### Scalability of the call setup delay in regard to the number of concurrent sessions

The scalability of MARCH will depend on the ability of CSs ability to simultaneously execute proxy modules and manage these proxy modules. The former depends of the functionalities of the proxy modules, their implementation, and the capabilities of the hardware. Thus is not possible to generalize and the scalability has to be investigated on an individual basis as described in [38]. The latter depends on the delay introduced by the MAS to CS signaling messages and the instantiation and configuration of a proxy module. In this section we only present the scalability of MAS - CS management.

We measured the session creation latency varying the type of session and the number of simultaneous sessions. The experimental setup used was identical to the one described above, i.e. using one proxy in the *proxy path*. For each type of proxy, we run a series of client requests at the MCE, and measure the time it takes to establish the session. For each point, a fixed number  $X$  of sessions is first established on the overall system by generating  $X$  requests. Once these sessions are established, an additional request is generated, and the time to create this additional session is measured. This was averaged over one hundred measurements to obtain an average of the call setup delay for the session under that particular load. Thus, the results presented in Figure 5 show the average of the call setup delay under a given load.

This measurement gives an estimate of the overheads introduced by the framework, but do not take into account the influence of network propagation delay and bandwidth between the MAS, MCE and CSs. The performance of the system in a real situation is likely to be worst as the client (MCE) will typically be connected to a high-latency network.

As explained earlier, we are only concerned with the scalability of MAS - CS management. Therefore the results represent the situation where no data is flowing through the proxy modules, i.e. there are no stream processing overheads.

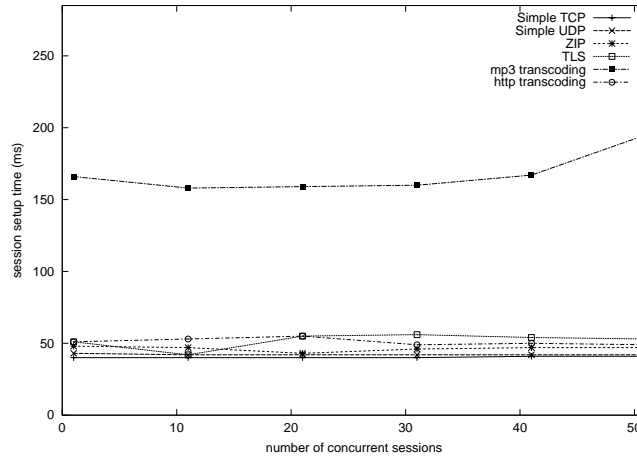


Figure 5: Call setup delay versus number of concurrent sessions

From Figure 5 it is clear that the latency to create MP3 transcoding sessions are significantly higher. This is due to the external process creation overhead. However, despite this difference in latency, for all modules, the cost of establishing additional sessions does not vary significantly. We have only shown the operation of up to 50 simultaneous sessions in Figure 5, as the upper bound is directly dependent on the amount of memory on the CS and other system resources on the MAS and CSs, which is not the focus of this investigation.

#### Scalability of the call setup delay in regards to the number of proxies installed

The other aspect of scalability in the MARCH framework is dependent on the number of *proxy chains* in a *proxy path* and the number of proxies per *proxy chain*. To investigate this, 15 Compute Servers were used to create *proxy paths* consisting of a variable number of *proxy chains* with different numbers of proxies per *proxy chain*. Then, we evaluate the dependence of the call setup delay on the number of proxies to be concatenated within each *proxy chain*, and on the number of *proxy chains* to be concatenated in each *proxy path*.

The experiment was done using the *null* proxy, a proxy who does not implement any functionality, the compute server's class cache was disabled, and the experiment is realized over an unloaded local area network. This experimental setup allows us to evaluate the scalability of the overall framework, in absence of any overheads associated with particular proxies or network conditions. Again, each point in Figure 6 represents the average of 100 measurements.

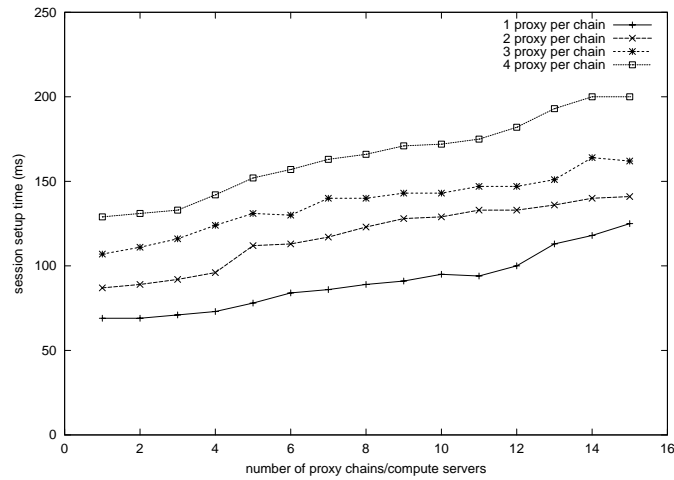


Figure 6: Call setup delay versus number of proxies in path

Figure 6 shows that the call setup delay increases linearly with the number of *proxy chains*. Furthermore, it shows that the relationship between call setup delay and the number of proxies per *proxy chain* is also linear, as the gap between each graph remains constant. The dependence of the call setup delay on the number of proxy chains per proxy path is due the overheads associated with configuration of the proxy modules, which are the dominant contribution to the call setup delay, as explained by the previous experiment.

## 5 Conclusions

In this paper we have presented a novel, server-centric architecture for adapting media content to suit the operational environment for heterogeneous devices and networks. The server-centric approach of MARCH exhibits several advantages over traditional static proxy solutions. First, by using proxy concatenation, MARCH allows service composition, i.e. to a compose proxy services individually tailored for each session characteristics. Secondly, the decision point of the framework being under the control of the content server administration, copyright issues are solved and proprietary content type can be accommodated. Moreover, the centralized decision point allows the MAS to distribute load evenly between compute servers.

The viability of the MARCH framework was demonstrated through a prototype implementation. It showed that the overhead associated with MARCH signaling was negligible compared with proxy download time and java overheads, over a local area network. Furthermore, it showed that the framework is scalable when a number of concatenated proxies have to be deployed for a session.

Therefore, we believe that the framework can play an important role in enhancing the quality of Internet services experienced by mobile users.

## References

1. P. Gunninberg and A. Seneviratne, "Services and architectures in the next generation internet using dynamic proxies," in *proc. of FTF99*, (Beijing, China), Dec. 1999.
2. S. Ardon, P. Gunningberg, Y. Ismailov, B. Landfeldt, M. Portmann, A. Seneviratne, and B. Thai, "Mobile aware server architecture: A distributed proxy architecture for content adaptation," in *proc. of INET2001*, (Stockholm, Sweden), June 2001.
3. L. Tennenhouse and D. J. Wetherall, "Towards an active networks architecture," in *proc. of Multimedia Computing and Networking*, (San Jose CA), Network Systems group, MIT, 1996.
4. M. Yarvis, A.-I. A. Wang, A. Rudenko, P. Reiher, and G. J. Popek, "Conductor: Distributed adaptation for complex networks," Tech. Rep. CSD-TR-990042, UCLA, Aug. 1999.
5. R. Keller, S. Choi, M. Dasen, D. Decasper, G. Fankhauser, and B. Plattner, "An active router architecture for multicast video distribution," in *proc. of IEEE Infocom 2000*, Mar. 2000.
6. Amir, S. McCanne, and R. Katz, "An active service framework and its application to real-time multimedia transcoding," in *proc. of SIGCOMM 98*, 1998.
7. M. Fry and A. Ghosh, "Application level active networking," *Computer Networks*, vol. 7, pp. 655-667, 1999.
8. "Web Intermediaries WEBI charter," <http://www.ietf.org/html.charters/webi-charter.html>. Accessed 01/06/2002
9. T.I. forum, "Internet content adaptation protocol," <http://www.i-cap.org/>. Accessed 01/06/2002
10. R. Mohan, J. R. Smith, and C.-S. Li, "Adapting multimedia internet content for universal access," *IEEE Transactions on Multimedia*, vol. 1, no. 1, pp. 104-114, 1999.
11. R. Han and P. Bhagwat, "Dynamic adaptation in an image transcoding proxy for mobile web browsing," *IEEE Personal Communications Magazine*, Dec. 1998.
12. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer, "Adapting to network and client variation using active proxies: Lessons and perspectives," *IEEE Personal Communications*, Aug. 1998.
13. "RabbIT proxy," [http://www.nada.kth.se/projects/prup98/web\\_proxy/](http://www.nada.kth.se/projects/prup98/web_proxy/). Accessed 01/06/2002
14. H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul, "An active transcoding proxy to support mobile web access," *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998.
15. T. W. Bickmore and B. N. Schilit, "Digester: Device-independent access to the World Wide Web," *Computer Networks and ISDN Systems*, vol. 29, no. 8-13, pp. 1075-1082, 1997.
16. Fox, S. Gribble, Y. Chawathe, and E. Brewer, "Adapting to network and client variation using active proxies: Lessons and perspectives," 1998.

17. Fox, I. Goldberg, S. D. Gribble, and D. C. Lee, "Experience with top gun wingman: A proxy-based graphical web browser for the 3com palmpilot," in *Proceedings of Middleware '98, Lake District, England, September 1998*, 1998.
18. Zenel, "A proxy based filtering mechanism for the mobile environment," Tech. Rep. CUCS-0xx-95, Computer Science Department, Columbia University, 1995.
19. "IBM software web application servers: Websphere transcoding publisher," <http://www.ibm.com/software/webservers/transcoding/>. Accessed 01/06/2002
20. "Avantgo 4.0," <http://avantgo.com/enterprise/products/avgo.html>. Accessed 01/06/2002
21. H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proc. of 1st ACM Conference on Mobile Computing and Networking*, (Berkeley, California), Nov. 1995.
22. "Performance enhancing proxies," <http://search.ietf.org/internet-drafts/draft-ietf-pilc-pep-05.txt> Accessed 01/06/2002
23. R. D. Silva, *PNUT: Protocols configured for Network and User Transmission*. PhD thesis, School of Electrical Engineering, University of Technology, Sydney, 1998.
24. Cohen, S. Rangarajan, and N. Singh, "Supporting transparent caching with standard proxy caches," in *Proc. of the 4th International Web Caching Workshop*, Mar. 1999.
25. Raman, R. H. Katz, and A. D. Joseph, "Universal inbox: Providing extensible personal mobility and service mobility in an integrated communication network," *Workshop on Mobile Computing Systems and Applications (WMSCA'00)*, Dec. 2000.
26. "W3C semantic web activity: Resource description framework (RDF)," <http://www.w3.org/RDF/>. Accessed 01/06/2002
27. J. Hjelm and M. Nilsson, "Position-dependent services using metadata profile matching," in *proc. of INET2001*, (Yokohama), July 2000.
28. S. Seshan, M. Stemm, and R. Katz, "SPAND:shared passive network performance discovery," in *proc. of 1st Usenix Symposium on Internet Technologies and Systems (USITS '97)*, (Monterey, CA), Dec. 1997.
29. S. Keshav, "Packet-pair flow control," *IEEE/ACM Transactions on Networking*, Feb. 1995.
30. Landfeldt, A. Seneviratne, P. Gunningberg, and B. Melander, "Network performance monitoring for applications using expand," *Engineering Transactions*, vol. 2, no. 3, 1999.
31. M. Portmann and A. Seneviratne, "The problem of end-to-end security for proxy-based systems," in *Proc. of Protocols for Multimedia Systems (PROMS 2000)*, (Cracow, Poland), Oct. 2000.
32. W. Ma, I. Bedner, G. Chang, A. Kuchinsky, and H. J. Zhang, "A framework for adaptive content delivery in heterogeneous network environments." Hewlett-Packard Laboratories, 2000.
33. R. de Silva, B. Landfeldt, S. Ardon, A. Seneviratne, and C. Diot, "Managing application level quality of service through TOMTEN," *Computer Networks*, vol. 31, pp. 727-739, 1999.
34. *Lame mp3 encoder homepage*, <http://www.sulaco.org/mp3/>. Accessed 01/06/2002

35. *Icecast mpeg-layer 3 streaming server*, <http://www.icecast.org/>. Accessed 01/10/2002
36. *Freeamp mpeg layer 3 player*, <http://www.freeamp.org/>. Accessed 01/10/2002
37. T. Dierks and C. Allen, "The TLS protocol," Request For Comments 2246, Internet Engineering Task Force, Jan. 1999.
38. R. Kehl, "Performance and scalability of real-time multimedia transcoding," Master's thesis, Computer Engineering and Networks Laboratory, ETH, Zurich, 2001.
39. W3C Composite Capability/Preference Profiles (CC/PP). <http://www.w3.org/TR/NOTE-CCPP/>. Accessed 01/10/2002
40. S.Ardon, "Intégration de l'utilisateur dans la gestion de la qualité de service pour des environnements hétérogènes", PhD Thesis, Université Pierre et Marie Curie, September 2002.
41. *Open Pluggable Edge Service (OPES)* <http://www.ietf.org/html.charters/opes-charter.html>. Accessed 01/10/2002
42. *RuleML homepage*. <http://www.dfki.uni-kl.de/ruleml/>. Accessed 01/10/02