Giving users an edge: A flexible Cloud model and its application for multimedia

Salekul Islam\textsuperscript{a,\textdagger}, Jean-Charles Grégoire\textsuperscript{b}

\textsuperscript{a} North South University, Dhaka, Bangladesh
\textsuperscript{b} Institut national de la recherche scientifique, Montréal, Québec, Canada

\begin{abstract}
We present the Edge Cloud architecture and explore how the user’s experience of the Internet is enhanced through its use, by moving desktop functionality within the network, in the Void environment. We explore how this model allows us to expand the range of benefits of using the Internet Cloud model, especially when it comes to media processing and user customization. A prototype that transcodes audio/video stream inside the Cloud has been implemented and presented. The prototype is deployed in two different environments: in a local lab and in the Amazon EC2 public Cloud. We also study the performance of the prototype by analyzing the inter-arrival jitter.
\end{abstract}

\section{1. Introduction}

Cloud computing is a collection of applications, hardware and system software designed to deliver services to end users over the Internet \cite{1}. The datacenter that deploys the necessary hardware and software is called a Cloud. Cloud computing offers a wide range of services, including storage and different modes of exploitation, i.e. Software/Platform/Infrastructure as a Service (S/P/IaaS) \cite{2}. Through virtualization technology, it is now possible to run not just an application, but also a full server inside the Cloud, thereby reducing the cost of the hardware base in a typical one service—one server model.

Beyond server virtualization, cloud computing has been used on the user side – mostly in enterprise environments – to provide remote access to desktops or specific applications, to give users access to a functionality which is too costly, insecure or inconvenient to deploy on their computers. Essentially, in user-side virtualization the application will run on a remote server while only the interface (or GUI) will run on the users’ computer. Historically this model was used to virtualize the whole desktop, in enterprise markets, but it now allows applications to be run from the Internet Cloud. SaaS provided through Cloud computing is one such example, as we have just mentioned.

The benefits of introducing virtual clients in the design of the future Internet architecture have hardly been explored to their full potential, and certainly not for the benefits of the general public. Whereas there is an increasing number of applications available to the general public through the Cloud, these applications are offered in isolation, through multiple providers, and introduce a fragmentation of the user’s environment as each has its own dedicated interface, making information sharing between applications a more complex task.

In a previous study, we have designed a next-generation Internet application support environment \cite{3}, which follows the key rationales of Cloud computing and Content Delivery Network (CDN) \cite{4}, and transfers the computing load from the client to the Cloud while bringing user-centered intelligence to the Internet edge. Latency, security and privacy, high bandwidth requirement for data-intensive applications are some of the major challenges Cloud computing is facing \cite{2}. Indeed, remember that the core of the Internet demands simplicity of operation to minimize latency and remains essentially dedicated to forwarding IP packets: any kind of additional inspection or caching of IP packets may severely increase the end-to-end delivery time.

We envision the Edge Cloud as a collection of many small Clouds: in addition to mostly application-centric Cloud infrastructures at remote, core-deep locations, multiple smaller, generic Clouds at the edge of the Internet are implemented in partnership with the edge ISPs \cite{5}. The Edge Cloud adds many benefits to the

\begin{thebibliography}{10}
\bibitem{1}...
\end{thebibliography}
existing Cloud Computing model including support for desktop virtualization, lower latency for the user, improved data transfer rates and traffic engineering, breaking the service vendor lock-in, fast access to local content, enhanced security and billing, etc. In this article, we focus on the architecture of Void, an Edge Cloud-based framework to support user-controlled coordination of Cloud-based applications and, as an example, its use in media processing. Void’s purpose is to support the Cloud-based virtualization of user applications typically running on the desktop, which require exchange of information and high data throughput. Unlike typical SaaS applications, Void supports desktop-like interactions between applications, and allows the user to create or manipulate composite applications.

Our focus on media processing is justified by the fact that multimedia, more specifically video streaming, is the leading traffic type in the Internet. According to a study [6], by 2013 the various forms of video (i.e., TV, VoD, Internet Video, and P2P) will exceed 90% of global consumer traffic. However, the absence of Adobe Flash Player in Apple’s iPhone or iPad, the existing Web browsers’ limited support for HTML 5 [7] video tag, or audio/video codecs’ patent issues [8] are some of the known difficulties the end users frequently face in receiving audio/video streaming. Media processing within Cloud facilities could overcome most of these hindrances and could provide a ubiquitous user experience irrespective of resources available at the users’ end (e.g. [9]).

The rest of the paper is organized as follows. Section 2 presents the state-of-the-art in Cloud computing and the architecture of the Edge Cloud. Section 3 presents the proposed architecture of Void, the potential services it provides and its benefits. A prototype of the Void, composed of Web servers, SIP clients and transcoding services is implemented. The prototype is deployed in a local lab environment and also in the Amazon EC2 public Cloud’s infrastructure. The implementation details are illustrated in Section 4. The performance analysis based on inter-arrival jitter analysis of both deployments of the prototype is presented in Section 5. The related work are being summarized in Section 6, with highlights on the originality of our model. Finally, Section 7 concludes the paper by summarizing our contributions and highlighting future studies.

2. Previous work

We begin with a review of the key concepts underlying our proposal, that is, a review of Cloud computing and a presentation of our Edge Cloud concept.

2.1. Cloud computing

The Cloud, available to the general public in a pay-as-you-go manner is known as the public Cloud. The Cloud implemented inside an organization for its own use is known as a private Cloud. A Cloud computing provider offers three features to the Cloud computing users: unlimited computing power on demand (or appearance thereof), adding resources only when the need increases (i.e. elasticity), with a pay-as-you-go model of billing (i.e. usage-based pricing) [1].

Fig. 1 shows four types of Cloud services identified in [2]. Although, the services are shown in a layered format, they are not always implemented in such a stacked way. Storage-as-a-Service is the simplest service, and includes Internet-based storage, middleware, database facilities for the users. Infrastructure-as-a-Service (IaaS) provides an on-demand computing infrastructure – a complete, yet virtual computer infrastructure – via the Internet. IaaS could be deployed for the enterprise through a private Cloud (e.g. Eucalyptus Enterprise Edition [10]) or it could be hosted by a third-party (e.g. Amazon EC2 [11]) through a public Cloud.

Platform-as-a-Service (PaaS) provides a full or partial application development environment that users can access and utilize online. For example, the Google App Engine [12] enables users to build and host Servlet-based Web applications on the same systems that host Google’s own applications. With SaaS, a provider licenses an application to customers either as a service on demand, through a subscription, or in a pay-as-you-go model.

Fig. 2 shows a simplified Cloud ecosystem consisting of four layers [13]. Since Cloud users need a flexible infrastructure on demand, a number of virtual machines (VMs) are deployed over physical resources. A VM manager (VMM, e.g. Xen, KVM, VMware) is required to manage the VMs and provides simple primitives including start, stop, and suspend. The virtual infrastructure (VI) management provides primitives to schedule and manage VMs across multiple physical hosts while cloud management provides remote and secure interfaces for creating, controlling, and monitoring virtualized resources on an IaaS Cloud. Commercially available IaaS Clouds, such as Amazon EC2 and Eucalyptus, implement the VI management and the Cloud Management (partially) layers. A separate Cloud Interface (e.g. Amazon EC2 WS) layer might be implemented on top of the IaaS. Finally, PaaS could be considered as the first level of support for Cloud consumers.

2.2. Edge Cloud

We have proposed in earlier work [3] our Edge Cloud next-generation Internet architecture, shown in Fig. 3. The Edge Cloud introduces a new layer, which we call the surrogate, on top of the traditional (storage, computing) core Cloud services. This surrogate layer acts as a relay for the user to access, organize, provide and monitor different composite Cloud-based services, of diverse nature. For example a hybrid service could be composed of a PaaS and a IaaS. In the rest of the text, we use XaaS to represent any flavor of service: SaaS, PaaS or IaaS or a hybrid thereof. Note that the term, XaaS is also used in [14] to refer to similar Cloud-based services. We must also note that the term surrogate is also used in RFC 3040 [15] to address a different type of network node, unlike our use in this paper.

The Edge Cloud, with the introduction of surrogate layer, helps us move away from the enterprise-based, mostly data-intensive use of the Cloud to meet the wider needs of the general

![Fig. 1. Different types of Cloud services.](image1)

![Fig. 2. Ecosystem of Cloud computing.](image2)
public. In a nutshell, different generic, and definitely more media oriented application components (e.g. IMS client, P2P client), traditionally implemented in the end user’s machine would rather be implemented as XaaS, to be accessed and reused by different users.

In this work, we take this concept further and expose how we can extend the surrogate to integrate other activities of the user’s desktop. Beyond the simple separation of the user interface from the application, we support the creation of hybrid applications and take the most benefit from the availability of the edge’s storage and compute resources.

3. Cloud-based user-oriented applications (Void)

A surrogate, as developed above, can be used to construct user applications in the Cloud, in our original Void model. Fig. 4 shows the surrogate-based architecture of Void. Void hides the complexities and the diversities of different services and provides a unified, user-specific context to the user. From the users’ perspective, the surrogate is their specific gateway for receiving various services including unified communications, content-specific services (e.g. search, add, mash-up, etc.) and other fundamental services (e.g. storage). Beyond that, the surrogate is home to their own customized XaaS applications or deploy user-specific content. Note that the user considers Void as a form of personalized virtual desktop composed of services that is subscribed/allowed to enjoy. Therefore, Void aggregates only those services that the user is authorized to use.

To keep this exposé simple, Fig. 4 is not showing the complexities related to managing virtual machines for the Edge Cloud. Therefore, an infrastructure layer composed of computing and storage services is shown as the bottom layer. Similarly, we consider that access to remote services is done through specific APIs, or even a captive sandboxed environment to run SaaS client javascript code.

3.1. Different services through Void

The Edge Cloud implements a wide range of services in the XaaS fashion, with Void acting as the glue for the user. A large variety of building blocks can be offered to support user needs, including access to search engine or P2P sharing. Closer to our purposes, different media processing services (e.g. transcoding, redirecting and mixing audio/video streams) can be implemented inside Void, to support a multimedia communications client.

Void also provides the opportunity to develop a mash-up model [16,17], that is, to give the user the opportunity of creating her own set of services from a set of offered XaaS-based building blocks (or elementary services). Moreover, a user could deploy her own user-generated services provided that she has proper authorization and the service is executable on the infrastructure of the Edge Cloud. How mash-up and user-generated services could be supported in Void is outside the scope of this study and hence these options are not shown in Fig. 4. The interested readers may readily find in the literature examples of work on resource allocation and performance management (e.g. [9,18]).

3.2. User access

User access to remote – including Cloud-based – services is done in two ways, either through the ubiquitous browser, or with a dedicated application.

The first one is certainly the most favored way to access common desktop-like and streaming services. It however suffers from a number of limitations. Browser sandboxing is meant to prevent randomly downloaded malicious code from accessing data on the computer but also within the browser itself. Such security mechanisms may prevent operations required to support a remote service, e.g. automated data transfer, and complicate processing for the user.

Another issue is access to some system features which fall beyond the traditional realm of the browser, typically media interfaces such as camera and microphone.

In this latter case, we have seen the emergence of hybrid models where separate applications (or plugins) will interface with the browser to use it as a GUI. Such hybrids are however browser/platform specific and have restricted universality. In the worst case, they may result in a multiplications of interfaces for the user. We would prefer to operate within standard extensions of browsers.

3.3. Benefits of Void

Building XaaS-based portals through a surrogate has not been studied before and thus the full scale benefits of this idea are to be explored yet. However, from our work, we postulate the following:

1. The software running on the user’s computer is streamlined by pushing complexity, compatibility and integration issues into the Edge Cloud.
2. New services could be built through mash-up from existing elementary services.
3. An application may depend on and require secure and efficient access to other services, e.g., the P2P engine depends on the index engine for locating a requested content.
4. In the enterprise environment, depending on role, position and hierarchy, a user should be able to use a specific set of services (which are delivered through XaaS). The surrogate could be
used to maintain user-profiles which allow administrators to add new services or delete the existing ones from the user's portal without modifying the XaaS layer.  

5. The XaaS provider could deploy the surrogates and new value added services. The present XaaS models only focus on enterprise-centric services. However, there also exists a large market for personal services. However, the reinforcement of the benefits of the surrogate model, that is, the simplification of the user's platform by focusing its use on the interaction with the applications rather than running the application themselves, is quite important. While this is in trend with SaaS and "apps", this model has the added benefits of giving the user more control over her own data, as well as being open to personalized service creation.

4. Prototype implementation

As a proof-of-concept, we present a prototype implementation of a Void-based multimedia infrastructure integrating a Web server, a Session Initiation Protocol (SIP) [19] client and transcoding services. The implementation architecture is shown in Fig. 5. Our purpose, through this implementation, is to expose key issues of relay/processing overhead and performance to demonstrate the technical feasibility of our concept.

4.1. Technologies used

Inside Void the Apache Tomcat Web server [20], an open source servlet container, hosts the Web-based user interface. The Web server receives the users input through the GUI running on the Web client inside the UE. The user performs SIP operations through Web interfaces on a SIP client running within the Tomcat server. We have implemented a simple SIP client using Java APIs for Integrated Networks (JAIN) SIP and Session Description Protocol (SDP) [21] stacks. Our SIP client performs SIP registration to the SIP server, generates and forwards the SIP INVITE message in response to the user request. Moreover, it exchanges SDP offer-responses with the SIP server. The transcoder is implemented using the Java Media Framework (JMF) [22], a Java library that enables audio, video and other time-based media to be added to Java applications and applets.

The application/media plane is composed of a SIP server, audio/video capture devices and a streaming server. The SIP server functionalities have been implemented using SailFin [23], which is based on robust and scalable SIP Servlet technology, on top of the reference, Java EE-based, GlassFish Application Server [24], and thus essentially added SIP Servlet technology to GlassFish. The standard audio/video capture devices, i.e., microphone and webcam connected with the machine that hosts application/media plane functionalities are being used. The SIP server works as a terminating point of SIP REGISTER and INVITE requests.

JMStudio, a standalone Java application that uses the JMF 2.0 API, captures audio/video streams and transmits those captured media streams across the network. JMStudio could also be used for transmitting recorded audio/video media files.

4.2. Message sequence

An example of message sequence of our implementation is shown in Fig. 6. We assume that all links are secured. In the following we briefly explain different steps of this message sequence.

4.2.1. SIP registration (steps 1–6)

First, the user accesses the Web-based user interface hosted by Void through its Web server. Next, the user enters the necessary information (user identities, address of the SIP server, etc.) for completing the SIP registration. This information is transmitted to the SIP client and triggers the SIP registration. The SIP client sends SIP REGISTER message to the SIP server. Note that, for the sake of simplicity, we have omitted standard Authentication

![Architecture of Void](image-url)
and Key Agreement (AKA)-based authentication. On successful SIP registration, the SIP server sends a SIP 200 OK message to the SIP client. It is worthwhile to note that the UE or the Web server triggers the SIP registration and that neither ever maintain any SIP registration related information.

4.2.2. Service request (steps 7–12)

Next, the Web server dynamically creates Web pages (using servlet and JavaServer Pages (JSP)) that list available media services (i.e., audio/video clips or live stream). Some of the audio/video medias would be directly delivered to the UE and some of them would be intercepted by the surrogate for transcoding. The user selects the media she wants to receive and a corresponding Service Request message is sent to the SIP client through the Web server. The SIP client sends an INVITE message to the SIP-AS that contains the URI of the media service (e.g., sip:webcam@10.128.128.26:5060) and the initial SDP offer for this call. In this URI, webcam is the identity of the media service and 10.128.128.26 is the domain name of the SIP-AS. An example of INVITE message is shown in Fig. 7.

The SIP-AS maintains the available codec formats for each media service. On receiving the INVITE message, the SIP-AS

INVITE sip:webcam@10.128.128.26:5060;transport=udp SIP/2.0
Via: SIP/2.0/UDP 10.128.128.222:6760;
branch=z9hG4bKdca035bc96b3d2901d3c1b316d037ca
Max-Forwards: 70
From: salekul <sip:salekul@10.128.128.222:6760>;tag=sipnmedia.0
To: webcam <sip:webcam@10.128.128.26:5060>
Call-ID: ff7510f3f577b43b3c8b3d2e2d77e0@10.128.128.222
CSeq: 1 INVITE
Contact: salekul <sip:salekul@10.128.128.222:6760>
Content-Type: application/sdp
Content-Length: 262

v=0
o= 3507993131 3507993131 IN IP4 10.128.128.222
s=-
c=IN IP4 10.128.128.222
t=0 0
m=audio 5004 RTP/AVP 0 1
a=rtpmap:0 PCMU/8000
a=rtpmap:1 1016/8000
m=video 7008 RTP/AVP 25 28 26
a=rtpmap:25 CeI/90000
a=rtpmap:28 ns/90000
a=rtpmap:26 JPEG/90000

Fig. 7. Example of SIP INVITE message.
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.128.128.222:6760;
branch=z9hG4bKdca3066e90bbd2501dcc1b311d837cae
From: salekul <sip:salekul@10.128.222:6760>;tag=sip5media0
To: webcam <sip:webcam@10.128.128.5060>;tag=gkr5mwc42
Call-ID: ff7510f3f5077b4e3b4c8bbbe22d77e@10.128.122
CSeq: 1 INVITE
Content-Type: application/sdp
Server: Glassfish SIP_1.0.0
Content-Length: 208

v=0
o=-- 3507993125 3507993125 IN IP4 10.128.128.26
s=-
r=IN IP4 10.128.128.26
m=audio 0 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=rtpmap:1 1016/8000
m=video 6800 RTP/AVP 34
a=rtpmap:34 H263/90000

Fig. 8. Example of SIP 200 OK message.

extracts the supported audio/video codec offered in the SDP content of the message and looks for one/more common codec(s) for the requested media. If any match is found, it is added in the SDP Response sent to the SIP client through the SIP 200 OK message. In such case, the media is directly delivered to the UE and no transcoding is required. This scenario, although implemented in our prototype, is not shown in Figs. 5 and 6 due to its simplicity.

If no common codec is found, the SIP-AS adds at least one codec in the SDP Response message. One such example is shown in Fig. 8, where the SIP-AS fails to find a codec supported by both parties and adds a=rtpmap:34 H263/90000 as the attribute of video codec.

Using the identity of the requested media (i.e. webcam), the media plane retrieves the URL and returns it back to the SIP-AS. The URL is forwarded inside a 200 OK message by the SIP-AS.

Note that to show the applicability of transcoding inside Void only one round of SDP offer/response is implemented although multiple rounds of offer/response messages could be exchanged between the SIP client and the media plane to agree upon a common codec. Moreover, in this specific scenario, the assumption that H.263 is not supported by the UE is a hypothetical case and not a practical assumption since it is a very common video compression standard indeed.

4.2.3. Direct media delivery (not shown in Fig. 6)

Direct media delivery (when transcoding is not required) to the UE is implemented in two ways: through HTML 5 and applet. When HTML 5 is used, the Web server dynamically creates an HTML 5 Web page by inserting the service URL inside a audio/video tag. This HTML 5 page is rendered to the UE. Thus, the user has the opportunity to start the media streaming with a navigation (i.e. pause, stop, volume control, etc.) capability. Media streaming and navigation opportunities are built into the latest version of HTML 5 [7]. However, at the time of this writing, not all browsers support these facilities.

Since we assume a simple UE with minimum software support, we are not using the Flash Media Player for playback-ing audio/video at the users platform. Alternatively, an applet that could display RTP media stream has been created. The applet is downloaded at the UE in on-demand fashion. The service URL (received inside the 200 OK message) is embedded inside the applet as the source of the RTP stream. Thus, the RTP stream could be viewed at the users' platform even in absence of the Flash Media Player.

4.2.4. Transcoded media delivery (steps 13–17)

For the transcoded model, the service URL is the UDP/RTP address to which the streaming server broadcasts the media stream. This UDP/RTP address is used as the input for the program which transcodes the input audio/video stream into another audio/video codec and possibly also changes other properties, e.g. sample rate, video resolution, etc. Next, the output UDP/RTP stream is forwarded to the UE. An applet is downloaded at the UE in on-demand fashion. The output stream of the transcoder program is embedded inside the applet as the source of the RTP stream. Thus, the transcoded RTP stream can always be viewed at the users’ platform.

Given that HTML 5 supports audio/video type tags, a UDP/RTP streaming address could be placed in the value of the audio/video type tag. The HTML 5 aware browsers only support limited number of video codecs [8] (e.g. H.264). On the other hand, our media streaming is based on JMF, which does not support any codec that is also understandable by the HTML 5-aware browsers. Hence, we could not take full advantage of HTML 5-based media streaming. The future versions of the browsers are expected to enhance the domain of the supported codecs. In that case, our prototype could be implemented without downloading the applet viewer onto the users’ platform.

5. Performance analysis of transcoding

We have studied the prototype of Void with a focus on the transcoding function in two different environments: our lab and using the Amazon EC2 infrastructure. A summary of different test results are given in Table 1. We have tested our prototype for transcoding both audio and video streams, which are denoted with A and V in Table 1. The source stream is generated either from a previously saved file or from a live stream captured by the webcam or microphone.

5.1. Lab tests

In the lab setup, a widely used network link emulator, the Anue Systems Network Emulator [25], has been deployed to introduce packet propagation delays on a specific link. We could introduce desired level of delays on the links that connect the user and the application/media plane with the Void. Hence, the scenario where the end user, the Void and the application/media plane are located in different networks is modeled. This delay is shown as ping time in Table 1, where the ping time (which is a roundtrip time) equals to double the delay time that the network emulator introduces.

5.2. Testing in Amazon's Cloud

To test our prototype implementation in the public Cloud, we have deployed the Void inside the Amazon Elastic Compute Cloud (Amazon EC2). First, we have launched a publicly available Amazon Machine Image (AMI) of Microsoft Windows Server and connected to it using a remote desktop protocol (RDP)-based interface. Next, we have installed and launched the Apache Tomcat Web server inside this instance. Through the Web interface of Tomcat, we have deployed the prebuilt Web application ARchive (WAR) file of our Void implementation.

We have tested the Void implementation at two different deployment regions (i.e., ec2-east and ec2-west) of the Amazon EC2 infrastructure. We have also measured the ping time (shown in Table 1) to the Amazon EC2 Tomcat Web server that hosts Void.

5.3. Jitter analysis

To assess the performance of the transcoder, we measure the jitter for different cases. Formally, jitter is defined as a statistical variance of the Real-time Transport Protocol (RTP) data packets'
inter-arrival time. The varying transit times are not important for non-real-time data packets (e.g., downloading a file using P2P client), however they play a key role in transmitting a stream of real-time data. Jitter is the parameter that gives us an estimation on how large the inter-arrival time differences are. For real-time streaming, the quality of the received stream is degraded if the inter-arrival jitter is high.

5.3.1. Jitter measure

We have used Wireshark [26], a widely used network packet analyzer, to capture network packets. Wireshark not only displays the captured packet data at different levels of detail, but also analyzes different network parameters from the captured packets. In this experiment, we have also used Wireshark to measure the inter-arrival jitter of RTP streams.

Wireshark calculates jitter according to RFC 3550 [27], which requires the RTP timestamp and the arrival time (both should be in same unit). The RTP timestamp is based on the sampling frequency of the codec. 8000 and 90000 are the sampling frequencies in most audio and video codecs, respectively. The inter-arrival jitter \( J \) is defined to be the mean deviation (smoothed absolute value) of the difference \( D \) in packet spacing at the receiver compared to the sender for a pair of packets. If \( S_i \) is the RTP timestamp from packet \( i \), and \( R_i \) is the time of arrival in RTP timestamp units for packet \( i \), then for two packets \( i \) and \( j \), the difference \( D \) may be expressed as:

\[
D_{i,j} = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i). \tag{1}
\]

The inter-arrival jitter should be calculated continuously as each data packet \( i \) is received, using this difference \( D \) for that packet and the previous packet \( i - 1 \) in order of arrival (not necessarily in sequence), according to the formula:

\[
J_i = J_{i-1} + \frac{|D_{i-1,i} - J_{i-1}|}{16}. \tag{2}
\]

Using Wireshark, we have captured both the source (i.e. outgoing stream of the streaming server at media plane) and the client (i.e. incoming stream at the UE) audio/video streams. Wireshark is also used in analyzing these streams and calculating delta (difference \( D \) and jitter \( J \) values. Jitter analysis for audio/video streams transcoding in the lab and Amazon EC2 Cloud settings are shown in Table 1. In the following, Figs. 9–16 have been generated directly by Wireshark. They show the evolution of the value of the jitter as computed with (2) over the course of a 30 s experiment.

5.3.2. Audio streaming

On receiving the DVI/RTP audio stream from the media plane’s streaming server, the Void transcodes the received stream to GSM/RTP audio stream and transmits it to the UE. Figs. 9 and 10 (Table 1, case 1) show the jitter for audio file stream from the lab deployment at source and client ends, respectively.

Figs. 11 and 12 (Table 1, case 3) show the jitter for audio file stream from the Amazon EC2-East Cloud deployment at source and client ends, respectively.

5.3.3. Video streaming

On receiving the H.263/RTP video stream from the media plane’s streaming server, the Void transcodes the received stream to JPEG/RTP video stream and transmits it to the UE. Figs. 13 and 14 (Table 1, case 6) show the jitter for live-captured video stream from the Amazon EC2-West Cloud deployment at source and client ends, respectively.

Figs. 15 and 16 (Table 1, case 8) show the jitter for live-captured video stream from the Amazon EC2-East Cloud deployment at source and client ends, respectively.

5.3.4. Comments on the measured jitter

In the following, we have summarized our observations on the analysis of the jitter data:

1. The measured jitter at the source end is in the range of few 100 μs (in the figures, microsecond is denoted by \( \mu s \)) to 5 ms, which is significantly less than the jitter at the client end (i.e., in the range of 5–30 ms). This difference is quite logical since the audio/video streams are generated at the source end and those streams go through transcoding inside the Void before they reach the client end.

2. If we compare the respective jitter of audio and video streams at the source end, we find that the jitter of audio streams (i.e. 5–10 ms) is less than the jitter of video streams (i.e. 5–30 ms). Table 1 gives us the evidence that a video stream generates many more packets than an audio stream. Hence, the transcoder has to deal with a large number of packets. Moreover, delays introduced during propagation may affect more packets.

3. We also compare the jitter measured from our lab experiments with the jitter measured from the Amazon EC2 deployments. During the lab experiment, the streams travel a very short distance in a closed network without any arbitrary delay or buffering except for the fixed delay introduced by the delay box. On the contrary, when the Void is implemented inside the Amazon EC2, the stream travels through the public Internet.

### Table 1

<table>
<thead>
<tr>
<th>Case</th>
<th>Deployment location</th>
<th>Stream type</th>
<th>Ping time (ms)</th>
<th>Measured end</th>
<th>Max delta (ms)</th>
<th>Max jitter (ms)</th>
<th>Mean jitter (ms)</th>
<th>Total RTP packets</th>
<th>Lost RTP packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lab with delay box</td>
<td>File, A</td>
<td>50</td>
<td>Source</td>
<td>63.63</td>
<td>0.57</td>
<td>0.11</td>
<td>501</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Lab with delay box</td>
<td>File, V</td>
<td>50</td>
<td>Source</td>
<td>461.98</td>
<td>51.83</td>
<td>2.08</td>
<td>716</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Amazon EC2</td>
<td>File, A</td>
<td>35</td>
<td>Source</td>
<td>66.57</td>
<td>0.82</td>
<td>0.17</td>
<td>623</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Amazon EC2</td>
<td>File, V</td>
<td>35</td>
<td>Source</td>
<td>147.97</td>
<td>27.93</td>
<td>6.38</td>
<td>624</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Amazon EC2</td>
<td>Captured, A</td>
<td>90</td>
<td>Source</td>
<td>135.40</td>
<td>35.01</td>
<td>26.91</td>
<td>662</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Amazon EC2</td>
<td>Captured, V</td>
<td>90</td>
<td>Source</td>
<td>80.79</td>
<td>4.76</td>
<td>4.27</td>
<td>663</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Amazon EC2</td>
<td>Captured, A</td>
<td>35</td>
<td>Source</td>
<td>135.06</td>
<td>23.53</td>
<td>17.38</td>
<td>539</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Amazon EC2</td>
<td>Captured, V</td>
<td>35</td>
<td>Source</td>
<td>157.06</td>
<td>2.96</td>
<td>0.41</td>
<td>347</td>
<td>0</td>
</tr>
</tbody>
</table>

Performance analysis of a transcoder deployed in the Edge Cloud.

### Figs. 9–16

Wireless analysis of transcoding in the lab and Amazon EC2 Cloud deployment at source and client ends, respectively.

### Table 1

<table>
<thead>
<tr>
<th>Case</th>
<th>Deployment location</th>
<th>Stream type</th>
<th>Ping time (ms)</th>
<th>Measured end</th>
<th>Max delta (ms)</th>
<th>Max jitter (ms)</th>
<th>Mean jitter (ms)</th>
<th>Total RTP packets</th>
<th>Lost RTP packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lab with delay box</td>
<td>File, A</td>
<td>50</td>
<td>Source</td>
<td>63.63</td>
<td>0.57</td>
<td>0.11</td>
<td>501</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Lab with delay box</td>
<td>File, V</td>
<td>50</td>
<td>Source</td>
<td>461.98</td>
<td>51.83</td>
<td>2.08</td>
<td>716</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Amazon EC2</td>
<td>File, A</td>
<td>35</td>
<td>Source</td>
<td>66.57</td>
<td>0.82</td>
<td>0.17</td>
<td>623</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Amazon EC2</td>
<td>File, V</td>
<td>35</td>
<td>Source</td>
<td>147.97</td>
<td>27.93</td>
<td>6.38</td>
<td>624</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Amazon EC2</td>
<td>Captured, A</td>
<td>90</td>
<td>Source</td>
<td>135.40</td>
<td>35.01</td>
<td>26.91</td>
<td>662</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Amazon EC2</td>
<td>Captured, V</td>
<td>90</td>
<td>Source</td>
<td>80.79</td>
<td>4.76</td>
<td>4.27</td>
<td>663</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Amazon EC2</td>
<td>Captured, A</td>
<td>35</td>
<td>Source</td>
<td>135.06</td>
<td>23.53</td>
<td>17.38</td>
<td>539</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Amazon EC2</td>
<td>Captured, V</td>
<td>35</td>
<td>Source</td>
<td>157.06</td>
<td>2.96</td>
<td>0.41</td>
<td>347</td>
<td>0</td>
</tr>
</tbody>
</table>
and therefore the propagation delay is not constant throughout the experiment. Hence, the inter-arrival time of data packets changes arbitrarily. Table 1 records the mean jitter for both types of experiments, which clearly shows that the mean jitter are less in the lab experiments than in the Amazon EC2 deployment.

6. Related work

Since Cloud computing is a vast area of research, we limit our discussion on work related to three key concepts of this paper: Edge Cloud, desktop virtualization through Cloud computing and media processing inside Cloud.
6.1. Edge Cloud

Integrating the virtues of CDN [4] and Cloud computing [1], the Edge Cloud brings the computing facilities closer to the user at the edge and implements storage at the edge. Compared to CDN, Cloud computing providers offer budget rates for Internet accessible data storage and delivery through their Cloud storage service. MetaCDN [28], a low cost, high performance content delivery system exploits these storage resources by building an overlay to federate multiple storage providers and then redirecting a request for content to the most appropriate replica to ensure good performance. The Edge Cloud is composed of smaller scale distributed Clouds while MetaCDN essentially piggybacks on top of the existing infrastructures. MetaCDN is being developed only for content distribution, and hence does not have any notion of client virtualization.

6.2. Desktop virtualization

Desktop as a Service (DaaS) [29], a virtual desktop paradigm, has evolved whereby desktops would be delivered as a service from a private Cloud (also referred as Desktop Cloud). There are many benefits to the DaaS model, including end-point ubiquity, subscription-based licensing, multi-tenancy, single instance management, etc. DaaS allows users to opt out of the costs or risks of owning and managing physical resources while adding the benefit of accessing desktops and desktop applications from any device at any location. There are several protocols used to access remote desktops (e.g., Microsoft’s Remote Desktop Protocol [30]) enabling desktop virtualization. Transforming legacy systems into desktop Cloud and proper capacity provisioning techniques are presented in [31]. The proposed Void model goes beyond the state-of-the-art desktop virtualization through Cloud computing; it integrates the surrogate and most importantly integrates all provisions for creating hybrid XaaS Cloud services.

6.3. Multimedia

An architecture for Cloud Multimedia Platform that can be used across different applications has been presented in [32]. A distributed computing framework, Hadoop [33] has been used to provide Platform as a Service (PaaS) type services for large scale data processing and storage applications. Another Hadoop-based audio/video media transcoding model is implemented in [34] to analyze different use cases. The results show the suitability of media transcoding inside Cloud that provides low cost streaming, such as the HTTP Live streaming scenario. Vmoox [35], a cross-platform video publishing solution for mobile devices, allows web publishers to distribute their video content (through real time transcoding) on mobile devices that is not equipped to play Adobe Flash contents. Note that Vmoox’s transcoding back-end could be implemented using SaaS Cloud services. An implementation of this model is presented in [36], that delivers transcoded Adobe
Flash content to Apple iPhones in real-time with the insertion of randomly selected commercial content.

6.4. A new dichotomy

The popularity of the client–server model for service offerings has been growing with the availability of ever cheaper access bandwidth and sustained the growth of Cloud-based applications. Also, the simplification of the user platform, often in conjunction with Cloud-based services, can be associated with the popularity of the “app” model of application deployment on smartphone platforms and its ingestion into the desktop model.

The client–server model has been explored extensively, including for the support of an application terminal, where some functionality can remain in the Cloud, possibly to support integration of different frameworks, based on different protocol bases, such as HTTP and SIP. The usual dichotomy is client-based integration or server-based integration. We have proposed instead here a three-tier model which gives the user more control over services and data—ownership need not be relinquished, or, at least, not to the application provider and it should remain possible to avoid its replication across different services.

Unlike other work, Void has the added benefit of involving the support of the local ISP, which facilitates integration of key functionalities such as security, mobility and QoS. This is however the focus of further work.

Our experiments have also established that the addition of our tier does not unduly penalizes overall application performance and confirmed the usefulness of the model.

7. Conclusion

We have proposed a new dichotomy between the user and Cloud-based services by introducing an Edge-based computation, coordination and storage facility. This expands the range of applications that can be migrated away from the user’s computer, while avoiding inconveniences of loss of control or data replication, over the currently prevailing Cloud SaaS model.

We have shown that this model is extremely useful for the support of multimedia applications, of diverse nature. We have implemented a testbed to study and analyze the performance issues arising from the use of our concept for media-based communications and shown that the benefits we projected can be achieved without significant performance loss.

This model offers a new role for the ISP—or an extension of his role as CDN provider, as well as an extended contract with the user which can go to guaranteed performance and support for mobility. These issues will be explored in further work.

References


Salekul Islam is an Assistant Professor of the Electrical Engineering and Computer Science Department of North South University, Bangladesh. He worked as an FQRNT postdoctoral fellow at INRS, a constituent of the Université du Québec. He has a Bachelors degree from Bangladesh University of Engineering and Technology in Computer Science and Engineering, a Masters degree and a Ph.D. degree both in Computer Science from Concordia University, Canada. His research interests are in the design, analysis and validation of protocols for telecommunication networks and secure multicast.

Jean-Charles Grégoire is an associate professor at INRS, a constituent of the Université du Québec with a focus on research and education at the Masters and Ph.D. levels. His research interests cover all aspects of telecommunication systems engineering, including protocols, distributed systems, network design and performance analysis, and more recently, security. He also has made significant contributions in the area of formal methods.