Computer Networks 56 (2012) 1763–1781

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

When Ambient Intelligence meets the Internet: User Module framework and its applications

Yu Lu [a,b,*], Mehul Motani [b], Wai-Choong Wong [b]

[a] Graduate School for Integrative Sciences and Engineering, National University of Singapore, Singapore
[b] Department of Electrical and Computer Engineering, National University of Singapore, Singapore

ARTICLE INFO

Article history:
Received 8 June 2011
Received in revised form 27 December 2011
Accepted 27 January 2012
Available online 4 February 2012

Keywords:
User centered design
User modeling
Computer networks
Protocols

ABSTRACT

Advances in Ambient Intelligence (AmI) technologies, when combined with user-centric computing, present major opportunities for building new communication pathways between end-users and traditional computer networks. We propose to explicitly take the end-user into account by defining a new functional module called the User Module across the layers of the Internet protocol stack. This new User Module empowers end-users to improve network performance and enhance the end-user Quality of Experience (QoE) based on their interaction activities. To successfully capture the significant interaction information of end-users, we leverage on the well-established Human Information Processing approach to build the end-user model. Subsequently, we present two exemplary User Module applications addressing the Hypertext Transfer Protocol (HTTP) and the Transmission Control Protocol (TCP), respectively. Both applications illustrate how the User Module operates to effectively improve the underlying network performance and eventually enhance the end-user QoE.

1. Introduction

As an effective technique for multiplexed utilization of interconnected networks and their hosts, today's Internet does not explicitly take into account end-users in its basic infrastructure. It is well-known that the 7-layer Open Systems Interconnect (OSI) model [1] and the 5-layer Internet protocol stack [2] provide layered architectures that partitions complicated network related tasks into different layers. Each layer has specific features and functionalities with peer interactions at equivalent layers across the networks. Both the traditional OSI model and the Internet protocol stack have played prominent roles in the success of the modern computer networks. Many fundamental and respected principles [3] are implemented in the Internet such as packet switching for multiplexing, end-to-end arguments for defining communication protocols [4] and global addressing for routing the datagrams. One of the design principles is that the Internet serves as the communication medium between two hosts that desire speaking to each other [5,6]. With such host-centric vision, both the 7-layer OSI model and the 5-layer Internet protocol stack simply regard the end-user and the network device as one entity, and inevitably ignore the end-user's presence, behaviors and interaction activities.

The AmI paradigm [7,8] is characterized by the intelligent environment and systems with prime emphasis on user-centered service, context-awareness and transparency. In other words, the AmI system aims at having "the system adapt to its users" rather than the other way round. Most AmI systems mainly concentrate on building the embedded systems for integration of various electronic devices and sensors into the environment for some specific application scenarios. To facilitate long distance data transmission for these AmI systems, the Internet and its underlying protocol stack has served as the default backbone data communication carrier. Nevertheless, few prior works con-

* Corresponding author.
E-mail addresses: lulu@nus.edu.sg (Y. Lu), motani@nus.edu.sg (M. Motani), elewwcl@nus.edu.sg (W.-C. Wong).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2012.01.019

sider introducing the AmI concept into the Internet protocol stack design to further enhance the Internet as a user-centric, intelligent, and interactive communication network.

As indicated earlier, the traditional OSI model and the Internet protocol stack, which emphasize host-centric design principle, do not adequately fulfill the requirement of our AmI vision. In our AmI vision, the interaction between the end-user and the underlying networks occupies a most crucial position in the user-centric and context-aware Internet design. Therefore, we propose to explicitly define a new functional module, called the *User Module*, across the existing layers as shown in Fig. 1. The newly proposed User Module aims to smoothly integrate end-users into the established Internet protocol stack, accordingly empower end-users to improve network performance based on their states and interaction activities, and eventually enhance end-users' subjective perception of network performance, namely the QoE.

The User Module operates under the client–server architecture and interacts with the underlying networks on the basis of the end-user state. The end-user state, in this work, refers to any conditions of Internet end-user that relate to the underlying networks and its applications, which covers the end-user presence, behaviors and interaction activities. Under the User Module architecture, several end-user models for different Internet services are built to derive the two defined end-user states in Section 4. We then present two exemplary User Module applications utilizing the built end-user models: the first HTTP case effectively reduces the end-user perceived latency in Web browsing and the unnecessary network traffic through adjusting the HTTP protocol in the application layer; the second TCP case dynamically allocates the limited bandwidth resource to enhance the end-user QoE through tuning the TCP protocol in the transport layer. Adjusting different layers and protocols usually improves underlying network performance and the end-user QoE from different aspects, and in this paper, the User Module mainly works with the upper two layers, i.e., the application layer and the transport layer. More ambitiously, all the existing layers can be involved in the proposed User Module framework as shown in Fig. 1. This paper makes three main contributions:

(1) Our AmI vision motivates a new user-centric network design through augmenting the traditional layered network architecture, and the proposed User Module explicitly incorporates end-users into the existing Internet protocol stack.
(2) We design and implement a concrete User Module framework consisting of three key functional subsystems. Based on such framework, we present how the human–computer interaction and cognitive psychology knowledge can be introduced and employed for building typical and reliable end-user models.
(3) Two exemplary and practical applications of the User Module, namely the HTTP and the TCP cases, are provided to improve the underlying network performance and enhance end-user QoE.

The outline of this paper is as follows. We introduce related work in Section 2. In Section 3, we propose the User

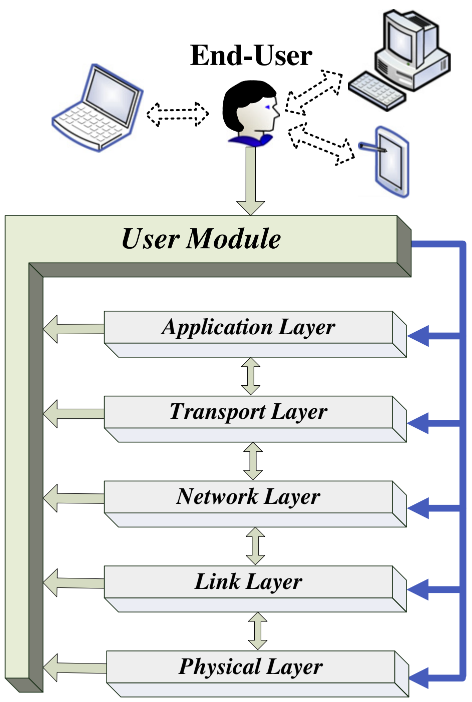


Fig. 1. User Module with the established network infrastructure.

Module architecture and introduce each of its main building blocks. In Section 4, we construct the model for end-users on the basis of the Human Information Processing approach. In Section 5, we present the first User Module application with comprehensive experimental results. In Section 6, we describe the second application under the same User Module architecture, and adopt the structured approach to assess the end-user QoE. Finally, we discuss possible future work and conclude in Section 7.

2. Related work

The AmI draws a blueprint for building new digital environments that adapt to end-user needs. With such a human-centric computing vision from both the technical and the social perspectives, there have been tremendous efforts [9–11] on designing AmI platforms. MIT has built a pervasive, human-centered computing environment in the Oxygen project [9]. The system deploys multiple embedded computational devices called Enviro21s (E21s) in offices, cars and homes to collect context information. With the hand-held devices called Handy21s (H21s) and the indoor location support, Oxygen's system can assist its users perform a group of tasks in their daily lives. Georgia Tech's researchers have designed an environment that can sense the inhabitants through a variety of sensing technologies in their "Aware Home" project [10]. One interesting Aware Home initiative called "Aging in Place" focuses on developing the technology and applications which enable senior adults to live independently in their homes. IBM has proposed the next-generation workspace solution in its "Blue Space" project [11], which integrated sensors, actuators, displays and wireless networks into the AmI environment. The workspace solution aims to increase productivity by deterring unwanted interruptions and facilitating communication among group members.

Most of the above-mentioned and related AmI systems emphasize on building the embedded intelligent environment, and the Internet with its underlying protocol stack only serves as the default long distance data communication medium. We aim to incorporate the AmI into the Internet infrastructure itself, and thus further enhance the Internet as a user-centric and context-aware intelligent communication network. Our AmI vision imposes many new challenges on the system-level design, where one of the most critical one is to understand the complicated interaction activities between the end-users and various Internet services.

Fortunately, the fields of Human–Computer Interaction (HCI) and cognitive psychology offer us numerous approaches to model the end-users and their interaction behavior. The Human Information Processing (HIP) approach [12] holds considerable promise to address how an end-user receives, stores, integrates and uses the information from the networks. The basic idea of the HIP approach is that the human is like a computer or a complex system that can be analyzed in terms of subsystems and their inter-relationships. Different HIP models have been developed to characterize or predict the end-user interaction activities and performance. The most widely known models include the Model Human Processor (MHP) proposed by Card et al. [13] and the Executive Process Interactive Control (EPIC) [14]. Both models assume that a series of discrete phases compose the information processing, and the output of one phase serves as the input for the next. The McClelland cascade model [15] considers that each phase is continuously active with continuous output values, where only partial information at each phase is transmitted to the next. Furthermore, some new approaches start to challenge and improve on the traditional HIP approach, such as the Situated Cognition [16] and the Cybernetic approach [17]. In this paper, our end-user model is built based on MHP, not only because it is the most widely known and established model, but more importantly, it offers an effective way to precisely define different states of end-users, which can be validated by specific communication conditions.

One of our main objectives of designing the User Module is to enhance the end-user QoE [18,19], which can be simply interpreted as the end-user's subjective perception on the qualitative performance of communication systems and applications. The ITU Telecommunication Standardization Sector (ITU-T) defines QoE as "the overall acceptability of an application or service, as perceived subjectively by the end user" [20]. Recently, particular attention is given to assess and measure QoE not only in terms of traditional Quality of Service (QoS) parameters [21], but a joint consequence of the communication context environment, the characteristics of the network service in use and the underlying network performance [22]. QoE is currently receiving immense interest from both the academic and industrial perspectives, and the progress on the techniques for enhancing and modeling the QoE will impact the user-centric network design and eventually benefit ordinary Internet end-users.

3. System overview

The system block diagram with the proposed User Module is illustrated in Fig. 2, and it operates under the traditional client–server architecture. The User Module consists of three core components: *User Interface subsystem*, *User Model subsystem* and *Control subsystem*.

3.1. User interface subsystem

The *User Interface subsystem* would directly interact with the individual end-user when it is operating at the client side. This subsystem always equips with some specifically designed User Interface and a variety of physical sensors to fulfill the following two main functions:

(1) Monitor and record the interaction activities and the context information between end-user and running Internet services. The interaction activities include both host-oriented and user-oriented behaviors, such as which application is currently displaying in the screen foreground, and whether the end-user is touching the mouse or keyboard. This subsystem is also responsible for collecting any valuable context information, such as end-user's location, identity and preference. Then it delivers all interaction and context information to the User Model subsystem in real-time for further processing. By leveraging on the ubiquitous sensing platform with intelligent physical sensors, the User Interface subsystem can operate and perform all its work in an invisible way.

(2) Besides automatically collecting the end-user interaction and context information in the background, the User Interface subsystem also provides end-users a direct and visible interaction service. For example, end-users can inform the protocol stack their states by simply pressing some matching buttons on a specially designed graphical User Interface, and the User Interface can show some significant network conditions to end-users through any user-friendly ways.

In short, the User Interface subsystem mainly undertakes the interaction information gathering task, and directly works with the end-user at the client side of the User Module.

3.2. User model subsystem

The *User Model subsystem* plays a key role in the User Module architecture, because it is the component for constructing, hosting and utilizing the individual user model and the group user model. The individual user model refers to the abstract data model for ascertaining the end-user presence, preference and interaction activities. In order to build an effective and accurate end-user model, HCI and the related cognitive psychology knowledge can be employed, which will be elaborated in Section 4. When more complex end-user states need to be identified, data mining and machine learning methods [23] can be introduced to

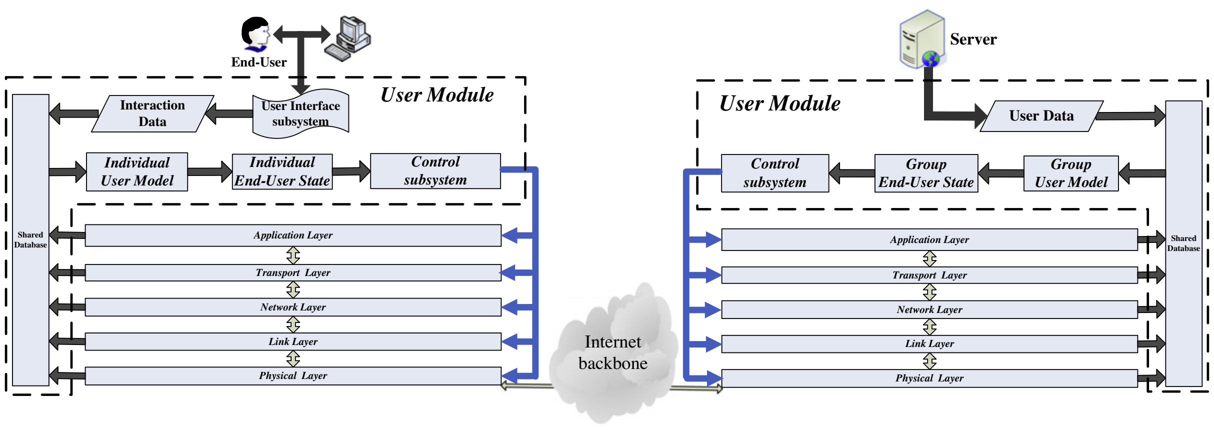


Fig. 2. User Module block diagram under the client–server architecture.

construct the corresponding end-user model, and some advanced reasoning approach, such as the ontology reasoning [24], can also be applied. The group user model usually presents at the server side, and it is mainly used to help enhance server performance and facilitate server batch processing.

The User Model subsystem mainly consists of two parts: a shared database and the built user model. The shared database is used to collect, retrieve and store important network condition from underlying layers. The network condition covers the current operating protocols, critical network configurations, and QoS parameters such as the real-time bandwidth consumption and the packet loss. Meanwhile, the interaction data from the User Interface subsystem are also delivered to the shared database at runtime. According to the built user model, the shared database then performs the task of data filtering to pick out all irrelevant data before sending the collected data. The built user model processes those collected data accordingly and generates the real-time end-user state, which will be promptly delivered to the Control subsystem. Note that the User Model subsystem can work at both the client side and the server side, where the corresponding built user model is the individual user model and the group user model, respectively. Accordingly, the output of the User Model subsystem, as shown in Fig. 2, can be the individual end-user state and the group end-user state.

3.3. Control subsystem

The *Control subsystem* is the component for directly interacting with the underlying network infrastructure. For different User Module applications, the Control subsystem may interact with different protocols and different layers, but its main objective is always to improve underlying network performance and enhance the end-user QoE. The Control subsystem could work at both the client side and the server side, receiving the individual end-user state or the group end-user state information. The Control subsystem essentially bridges the gap between the User Module and the underlying network infrastructure. In this paper, the User Module only works with the upper two layers, and how to enable the Control subsystem to interact with the lower layers of Internet protocol stack needs to be addressed in future work.

When interacting with the underlying network, the Control subsystem does not attempt to modify any internal structure of the existing network protocols and Internet system architecture. In most cases, the Control subsystem would only cautiously choose the proper parameters, which are usually accessible and adjustable in their corresponding protocols and layers, and then actively tunes these parameters in accordance with the pre-defined control rules. Thus it is relatively different from the traditional cross-layer design [25]. Cross-layer design always exploits the dependence between protocol layers to obtain performance gains and typically follows some basic structures, such as creating new interfaces or merging of adjacent layers [26], to share and exchange state information, while the Control subsystem focuses on actively managing the exposed parameters of the existing network protocols and configurations. Hence, in general, the integrity of the conventional Internet layered architecture and protocols can still be well maintained, when the User Module with the given Control subsystem is introduced and implemented.

The above-described three subsystems compose the core architecture of the User Module, and such design explicitly and smoothly incorporate the end-user into the Internet mainstream infrastructure.

4. Modeling the end-user

In order to build an effective User Model subsystem, we must first understand how the end-user interacts with the underlying networks, or more specifically the Internet services and applications on the networked host. As indicated earlier, HCI and cognitive psychology fields offer a variety of well developed frameworks and models to explain and describe the end-user interaction behavior. The HIP approach is one of the most successful methods to conceptualize how the end-user's mind works. The basic idea of this approach is that human behavior is a function of several ordered processing stages. In other words, the human is like a system that can be analyzed in terms of subsystems and their interrelation. Different architectures, such as the ACT [27] and the SOAR [28] models, hold great promise for the HIP approach, while the most widely accepted and well-known model is the Model Human Processor (MHP) proposed by Card et al. [13].

As shown in Fig. 3, MHP consists of three interacting subsystems: the Perceptual subsystem, the Cognitive subsystem and the Motor subsystem, and each with its own processors and memories. The Perceptual subsystem is equipped with sensors and associated buffer memories for collecting and temporarily storing the external information. The Cognitive subsystem accepts symbolically coded information from the memories of the Perceptual subsystem, and then decides on how to respond. Finally, the Motor subsystem carries out the response and takes action. The MHP models the information processing of the end-user as a sequential or parallel operation of these three MHP subsystems. Furthermore, the Rationality Principle and Problem Space Principle of MHP indicate that the end-user's behavior is based on rational activity, which means the end-user will not randomly and arbitrarily change from one state to another. Moreover, all rational activities serve to achieve the end-user's goals, given the task and external information and bounded by the user's knowledge and processing ability.

Based on the MHP and the two basic principles, we can define many different but reasonable end-user states to describe the complex interaction behavior between the end-user and the Internet services. In this paper, we define two general but significant end-user states:

(1) *User Communicating State* (*UCS*): Both the end-user's Perceptual and Cognitive subsystems are turned ON to acquire and process the information of the corresponding Internet application.
(2) *User Inactive State* (*UIS*): The end-user's all *three* subsystems are turned OFF with the corresponding Internet application, and thus there is no interaction between the end-user and that application.

The above-defined two states essentially depict the two typical engagement levels of the end-user with the Internet applications based on the three subsystems of MHP. For other situations and intermediate engagement levels, we simply define them as *Unidentified User State* (*UUS*). Note that the three defined end-user states can be applied to any Internet applications, such as the Web browser, where each Web page tab is considered as one single application. Each running application can be associated with only one end-user state: *User Communicating State*, *User Inactive State* or *Unidentified User State*.

From the definition of these end-user states, we know that monitoring and recognizing the status of the end-user's three MHP subsystems is the most straightforward way to identify the real-time end-user states:

- For the end-user Perceptual subsystem, MHP shows that the most important memories for the human perceptual processors are the Visual Image Storage and the Auditory Image Storage. Thus the User Module monitors the source of the visual and auditory information, or more specifically which application generates the visual or auditory output. On the other hand, the User Module should also verify whether the end-user perceives that visual or auditory output.
- For the end-user Cognitive subsystem, although many researchers attempt to model and build the cognitive architecture [29], it is still difficult to accurately differentiate its status. Fortunately, MHP demonstrates that the Motor subsystem follows the Recognize-Act Cycle of the Cognitive Processor. Thus through monitoring the Motor subsystem behavior, the User Module can estimate the status of the Cognitive subsystem.
- Within the end-user Motor subsystem, the arm-hand-finger system is considered as the most important actuator by MHP. Therefore, from observing the keyboard and the mouse input information, the User Module can deduce the specific application which the end-user's Cognitive subsystem is working on. Moreover, MHP takes the human vocal system as another actuator of the end-user Motor subsystem. Hence, it is also a significant clue to deduce which application is being processed by the Cognitive subsystem of the end-user.

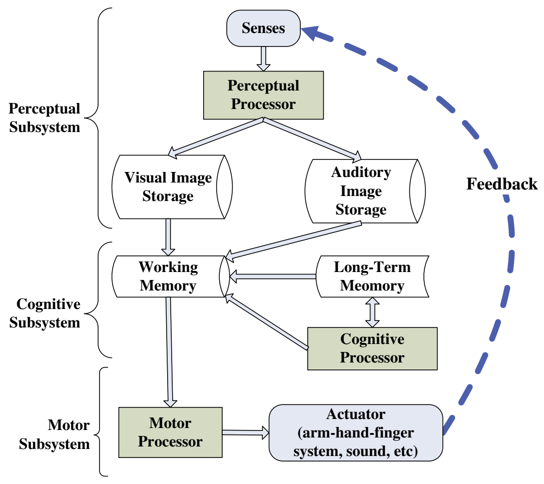


Fig. 3. Model Human Processor (MHP) framework.

Based on the above analysis of MHP and the newly defined end-user states, five *communication conditions* with the corresponding *Validation Criteria*, as shown in Table 1, need to be verified by the User Module. Note that for each specific application, the given communication conditions in Table 1 should be verified respectively according to the corresponding Validation Criteria.

We implement a User Interface subsystem specifically designed for detecting the five communication conditions in Table 1. For example, it can automatically monitor and collect valuable real-time information from the operating system, such as which application is displaying in the foreground of the screen and which application is receiving the mouse/keyboard input. Moreover, a common USB or a built-in Webcam, as shown in Fig. 4, can directly serve as a sensor to capture the end-user's open eyes and estimate the eyes-gaze direction, and thus can efficiently sense whether the end-user is watching the network host screen. The Open Source Computer Vision (OpenCV) library [30]

Table 1
Communication conditions and corresponding validation criteria.

	Communication Condition	Validation Criteria
(1)	The end-user is perceiving visual information	The corresponding application displaying in screen foreground
		The eye-gaze direction of the end-user towards host screen
(2)	The end-user is perceiving auditory information	The corresponding application generating audio output
		The end-user either wearing earphone or near speaker
(3)	The end-user generates mouse input	The corresponding application receiving mouse message
		The end-user touching mouse
(4)	The end-user generates keyboard input	The corresponding application receiving keyboard message
		The end-user touching keyboard
(5)	The end-user generates microphone input	The corresponding application receiving audio input
		The end-user near microphone

and the existing visual tracking algorithms [31,32], greatly facilitate building such a video-based eye-tracking system. Besides verifying the Validation Criteria in Table 1, the User Interface subsystem also records some important network conditions. For example, it can periodically sample the bandwidth consumption of each running Internet application from both the transport layer and the physical layer at the end-user side. To measure the bandwidth consumption in real-time, a third party driver called WinPcap [33] is employed in the User Interface to intercept packets flowing through the network adapter. All the collected interaction data of end-user and the network-related information are directly delivered to the shared database. Some functions have not been implemented for the current version of our User Interface subsystem such as sensing whether the end-user is sitting near the speaker, but a great deal of research has been done for solving such positioning problems and many RFID location sensing systems have been commercialized [34]. Our User Interface subsystem is mainly developed in Visual C++ under Microsoft.NET Framework on Win32 platform.

After implementing such User Interface subsystem, we can efficiently develop end-user models for various Internet services. Rather than covering a large number of Internet applications, we have chosen three representative Internet services with their corresponding applications for our user study: *Web Browsing*, *Live Multimedia Streaming* and *File Transferring*.

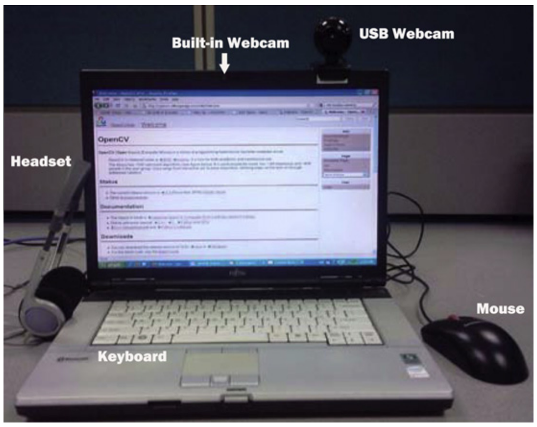


Fig. 4. Sensors in the User Interface subsystem.

- The primary purpose of Web Browsing is to bring the information on the Web Server to the end-user. In our user study, we select the popular Web browser: Mozilla Firefox.
- Live Multimedia Streaming provides live television or live radio service over the Internet. It always requires a minimum guaranteed bandwidth allocation, because the real-time video/audio programs are sensitive to fluctuations of the received rate. A Live Multimedia Streaming application called QQQTV is used in our user study.
- File Transferring refers to copying a file to or from a remote host over the Internet, and it is also one of the most utilized Internet services. We use CuteFTP, which is a widely-used file transfer application based on the standard File Transfer Protocol (FTP), in our user study.

Based on our implemented User Interface subsystem and the defined end-user states, we conduct a user study with multiple participants to determine the end-user models for each of the above-described application. A group of simple but effective end-user models are summarized in Table 2.

In Table 2, we see that different combinations of communication conditions for the corresponding Internet services indicate different end-user states, where “S” means satisfying the corresponding communication condition, “F” denotes failing to satisfy it, “X” means either of the above two options, and “n/a” indicates not applicable for that application. For example, when the User Interface subsystem has verified that the end-user is perceiving the visual and auditory information, i.e., the *communication conditions* (1) and (2) in Table 1, from QQQTV (live streaming application), then the User Communicating State can be directly associated with QQQTV regardless of whether the end-user generates the Mouse, Keyboard and Microphone input to QQQTV, i.e., the *communication conditions* (3), (4) and (5) in Table 1. On the other hand, when the User Interface subsystem has verified that the end-user is not perceiving any visual and auditory information from QQQTV and also not generating any Mouse and Keyboard

Table 2
The individual user models for different internet applications.

Internet Application	User State	Communication Condition				
		(1)	(2)	(3)	(4)	(5)
Firefox *(Web Browser)*	UCS	S	X	S	X	n/a
	UIS	F	F	F	F	n/a
QQQTV (Live Streaming)	UCS	S	S	X	X	X
	UIS	F	F	F	F	X
CuteFTP (File Transfer)	UCS	S	n/a	S	X	n/a
	UIS	F	n/a	F	F	n/a

S=Satisfy; F=Fail to satisfy; X=either S or F; n/a=not applicable.

input to QQQTV, then the User Inactive State can be associated with QQQTV regardless of the Microphone input condition. Furthermore, we categorize other possible combinations of communication conditions, which are not described in Table 2, into *Unidentified User State* (*UUS*). The end-user models in Table 2 work well for the given applications in most situations, and more complicated end-user models for some specific circumstances can be constructed through other cognitive psychology architectures and advanced reasoning approaches.

From the end-user perspective, the built end-user models based on the MHP can efficiently detect the User Communicating State and the User Inactive State for the given Internet services and applications in real-time. From the network perspective, it is also necessary and significant to build such models to differentiate the Internet end-user's communicating and inactive state. The main reason is that today's Internet still follows the traditional design principle that it serves as the communication medium between any two hosts that desire speaking to each other. In other words, Internet communication protocols with the layered architecture do not take into account the end-user state, and essentially conflate the dynamic end-user and the static network host into one simple concept, namely the Internet host. For example, Hypertext Transfer Protocol (HTTP) and Transmission Control Protocol (TCP) follow such traditional principle, and they have been widely used to support various Internet services, including the above-described Web browsing, Live Multimedia Streaming and File Transferring services. Such traditional design principle greatly decreases the complexity of today's Internet communication protocol design, but inevitably compromises the underlying network performance and accordingly causes many problems, such as the end-user QoE issue and the end-user mobility issue. Therefore, the built end-user models in Table 2 under the User Module architecture essentially provide Internet an efficient de-conflation solution to separate the end-user from the Internet host and recognize the two basic end-user states.

In order to introduce the identified real-time end-user states into the Internet protocol stack, the Control subsystem needs to make the necessary adjustment to the corresponding communication protocols in a way of selecting and tuning the proper parameters. To demonstrate this, we select two parameters in HTTP and TCP, namely the *persistent connection timeout* in HTTP and the *advertised window size* in TCP, and present two distinct exemplary User Module applications in the following sections, respectively.

5. The User Module application 1: HTTP

In the first User Module application, the Control subsystem mainly manages the *persistent connection timeout* in HTTP based on the derived end-user states. Such application could effectively reduce the end-user perceived latency in Web browsing and the unnecessary network traffic.

5.1. Problem description

Hypertext Transfer Protocol (HTTP) [35] is the de facto application-layer communication standard for transferring the Web pages. The persistent connection mechanism of HTTP/1.1, also called HTTP keep-alive, allows HTTP clients to send multiple requests over the same TCP connection. The persistent connection mechanism reduces network congestion from re-establishing TCP connections and conserves the host's CPU and memory usage. As a default function, persistent HTTP connection is widely implemented on both the browser and the server sides. HTTP/1.1 [35] specifies that "*servers will usually have some time-out value beyond which they will no longer maintain an inactive connection*", and "*the use of persistent connections places no requirements on the length (or existence) of this time-out for either the client or the server*". Clearly, HTTP/1.1 does not explicitly define the persistent connection-closing mechanism but suggests picking a proper timeout value for terminating the persistent connection. In the practical implementation of HTTP/1.1, a fixed timeout value is always imposed. The latest version 2.2.1 of the Apache HTTP Server employs 5 s, and the Microsoft IIS uses 120 s as their default timeout values. Improperly configuring the timeout value will easily degrade network performance. A small fixed timeout value causes low utilization of the persistent HTTP connection, and thus increases the end-user perceived latency as well as the Internet burden. Conversely, a large fixed timeout value will waste and even quickly exhaust limited Web server resource, which also results in long and unpredictable end-user perceived latency.

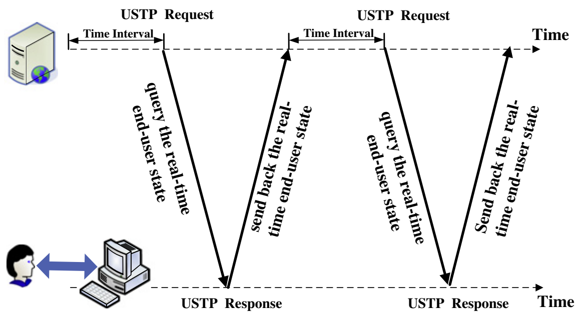
There is limited research work to optimally tune the persistent connection timeout value of HTTP to improve Web server performance: Faber [36] and Barford [37] indicate that the Web server should close the persistent connections once the client becomes inactive, but no specific approach has been provided. Mogul [38] proposes to give higher priority to the newly established connections, while Sugiki [39] suggests setting higher priority to the small RTT connections and prematurely terminate the ones with large RTT. However, none of these previous studies directly solves the main problem of HTTP persistent connection mechanism. *In a Web session, it is difficult for HTTP to discriminate between a persistent connection that is being used by the end-user and a persistent connection that is already in a long-term idle state.* Therefore, adopting the built end-user models under the User Module framework becomes a natural and effective way to address this problem. In order to transfer the derived end-user states information from the client side to the Web server side, it is necessary to first define the User State Transfer Protocol (USTP).

5.2. User State Transfer Protocol (USTP)

The *User State Transfer Protocol* (*USTP*) is used to deliver the real-time individual end-user state from the client side to the server side. The USTP assumes a reliable transport and in this case using HTTP persistent connection, which is essentially the TCP connection, as its underlying carrier. The USTP works under the client–server architecture and employs the request-response message exchange pattern. The overall workflow of the USTP is illustrated in Fig. 5:

(1) After the persistent HTTP connection becomes idle, the Web server side USTP program waits for a time interval and then initiates a request message to the client side via the existing persistent HTTP connection.
(2) Upon receiving the USTP request, the client side USTP program retrieves the real-time individual end-user state from the User Model subsystem of the User Module. The retrieved end-user state can be User Communicating State, User Inactive State or any other pre-defined end-user state.
(3) The client side USTP program encapsulates that real-time end-user state in an USTP response message, and sends it back to the Web server side.
(4) The Web server side USTP program receives multiple USTP response messages from different client sides. Then it delivers those end-user states and relevant information to the group user model subsystem for further processing.
(5) After some time interval, the server side USTP program repeats the request via the same persistent HTTP connection, if that connection still exists and remains in the idle state.

Theoretically, the USTP should adjust the time interval between consecutive requests according to the end-user's browsing behavior pattern. Based on the MHP theory, the end-user's browsing behavior primarily depends on his Cognitive subsystem, whose tasks involve learning, retrieving the facts from its long-term memory and acquiring the solution of the problem. Through practical user study and theoretical calculation, MHP shows that the cognitive processing rate has a wide range among different individuals because of their different processing capacities. For example, human reading speed ranges from 52 to 652 words per minute; in working memory, the decay parameter varies from 5 to 226 s. In other word, even for the same Web page, different end-users require different processing times and the variance magnitude can be several seconds or even larger. We simplify this cognitive diversity by using 7 s, the average decay parameter value of the human cognitive subsystem [13], as the time interval value of USTP. In the future work, the User Interface and the User Model subsystem could attempt to capture the end-user's individual cognitive capacity and roughly predict the processing time, then the system performance can be further improved.


USTP Request
Time Interval
USTP Request
Time Interval
Time
query the real-time end-user state
send back the real-time end-user state
query the real-time end-user state
Send back the real-time end-user state
USTP Response
USTP Response
Time
Fig. 5. The User State Transfer Protocol workflow.

Remark 1. The current USTP operates only in the simple condition, and a more complicated situation occurs when one or more intermediaries are present between the end-user and the Web server, such as when a proxy server is in use. Under such situation, a new version of USTP needs to be specified.

Remark 2. Transferring end-user state information to server side may raise security concerns. When necessary, some encryption protocols, such as the Transport Layer Security (TLS), could be adopted to prevent eavesdropping or tampering. Meanwhile, the User Module should also make every effort to process end-user information locally and avoid any unnecessary transmission between two ends.

The USTP is designed to provide a timely delivery of the individual end-user state from the User Module's client side to its server side. The current version of the USTP presumes a reliable transmission carrier and defines a simple invocation function *us_remote_request ()* on the server side. Hence, it is also applicable to other User Module applications that require the dissemination of the derived end-user states.

5.3. The control subsystem

Once the USTP response message successfully transfers multiple end-user states to the Web server side, the group user model can utilize the delivered information to help enhance server performance. For example, a group user model could classify all the delivered User Communicating States into several categories according to their duration time, and accordingly the Web server would provide the service differentiation to these fine-grained categories, e.g., shorter duration receiving higher priority. Since we mainly aim to reduce individual end-user perceived latency, we do not build the group user model for this application but simply move each delivered end-user state to the Control subsystem at the server side to continue processing.

After successfully obtaining the real-time end-user state, the Control subsystem at the server side adopts the following control rules:

(1) IF the User Inactive State arrives, THEN the Control subsystem immediately signals the application layer to *terminate* the corresponding HTTP persistent connection, i.e., setting the persistent connection timeout parameter to zero.
(2) IF the User Communicating State arrives, THEN the Control subsystem signals the application layer to *maintain* the corresponding HTTP persistent connection and *wait* for the next end-user state from the same client, i.e., setting the persistent connection timeout parameter to any large value.
(3) IF the Unidentified User State arrives and the Web Server is under the heavy-traffic situation, THEN the Control subsystem handles it as the User Inactive State. Otherwise the Control subsystem treats it as the User Communicating State.

The above control rules enable the inflexible HTTP persistent connection mechanism adapt to the end-user's real-time browsing behavior, and influence the end-user perceived latency as well as the network performance. Consequently, we conduct comprehensive experiments for assessing the performance gain from both the end-user and the network perspectives.

5.4. Experimental setup

5.4.1. Server-side implementation issues

We select the Apache HTTP Server in our experiment, as it is a popular open-source Web server. The current Apache HTTP Server 2.2 is configured by writing different Directives in its configuration files, and the HTTP persistent timeout value is set in the main configuration file by the KeepAliveTimeout Directive. The Apache HTTP Server places the KeepAliveTimeout Directive in its main configuration file apache2.conf and sets 5 s as its default value. However, any changes to the KeepAliveTimeout Directive can only be recognized by the server when it is started or restarted, because the Apache HTTP Server only reads and processes the main configuration files during its boot-up phase. Furthermore, the KeepAliveTimeout Directive can only simply set the same timeout value for all incoming HTTP requests, so different timeout values cannot be set for different HTTP connections which are initiated by distinct end-users. Since the existing Apache configuration mechanism and KeepAliveTimeout Directive cannot meet our requirements, we therefore disable the KeepAliveTimeout Directive and modify a small part of the Apache source code, where the Apache Server implements the HTTP persistent connection function. The newly modified Apache HTTP Server can adaptively adjust the timeout value according to the real-time end-user state, and it does not require any restart or reboot. Meanwhile, the newly modified Apache HTTP Server can also set different timeout values for different incoming HTTP requests, which are initiated by distinct end-users.

We employ the dynamic Web pages as the workload file in our experiment. Thus PHP code is embedded into the workload HTML files and interpreted by the PHP processor module. We install and configure the PHP 5.3.2 module on the modified Apache HTTP Server under the Linux 2.6.28. The average PHP processing time in the experiment is 50 ms, which also takes accounts for the time to access the database.

5.4.2. Client-side implementation issues

In order to emulate multiple end-users' Web browsing scenarios, the experiment requires a specific HTTP request generator to perform the following functions:

- The HTTP request generator can emulate the defined User Communicating State and User Inactive State. During the User Communicating State, the generator starts to make a HTTP request and after receiving the whole Web page from the server, waits for a certain time interval (end-user processing time) before it sends the next HTTP request. The generator repeats the above procedure until the User Communicating State ends. Then, during the User Inactive State, it simply keeps silent and stops sending HTTP request until the User Communicating State resumes.
- To simulate multiple concurrent end-users and create heavy-traffic condition, the HTTP request generator should be able to simultaneously open multiple sockets and initiate HTTP requests through each socket independently.

Existing popular Web workload generators, such as SPECweb2005 [40] and Surge [41] cannot fulfill both of these functions, thus we implement a new HTTP request generator. The new generator is written based on Libwww [42], which is a highly modular and flexible client side Web API for both UNIX and Windows (Win32) platform. We build the new HTTP request generator under the Linux 2.6.28 and all the code is written in C. The new HTTP request generator can open multiple sockets simultaneously, and control the HTTP requests on each socket. The next HTTP request can only be sent after receiving the last HTTP response and waiting for some manually defined time interval. Therefore, the newly built HTTP request generator can emulate multiple end-users generating HTTP requests concurrently, while any one of its open sockets simulates single end-user's Web browsing behavior by means of sequentially sending HTTP requests.

5.4.3. Experimental configuration

Our experimental hardware setup involves several hosts connected to the campus local area network (LAN). Each host is equipped with Duo Intel T7300 2.00-GHz processors, and a 2-GB RAM, and runs Linux 2.6.28. One of the hosts is selected as the Web server and runs Apache HTTP Server 2.2.15. Other hosts act as multiple end-users and keep generating HTTP 1.1 requests to the Web server. For different traffic load cases, each host simulates different numbers of end-users ranging from 30 to 300. The experiment topology, as depicted in Fig. 6, consists of multiple end-users and one Apache HTTP Server.

In the experiment, we adopt Dummynet [43] to emulate the practical Internet environment. Dummynet is a widely-used tool for enforcing queue and bandwidth limitation, delay and packet loss in the network experiment and test. We enable its delay function and set the configuration parameter to 100 ms for both directions of each link. So the Round Trip Time (RTT) between the client and the Web server is around 250 ms, which consists of the packet-propagation delays, PHP processing time and packet-queuing delays. We also set the packet loss rate to 1%, which is usually caused by the congestion and data corruption along the path of data transmission. The above conditions commonly exist in a Wide Area Network (WAN) as well as the last-hop wireless networks environments. Since optimizing the Web server's overall performance is out of the scope of this experiment, most of the configuration parameters of the Apache HTTP Server are kept at their default values. For example, the maximum number of connections can be processed simultaneously by the Apache HTTP Server is 256 in the experiment.

We consider two experimental conditions: *light-traffic condition* and *heavy-traffic condition*. Note that the heavy-traffic condition here is different from the server overload situation. The heavy-traffic condition means that the number of concurrent alive clients reaches the maximum number of allowable connections, while the overload situation indicates that the workload persistently exhausts some server resources, such as the CPU load or the server uplink bandwidth. Since system performance always becomes unstable under such overload situation, we do not consider it in our experiment and we suppose that the Web server employs some admission control schemes, such as in [44,45] to avoid overload situation. For the *light-traffic* condition, each host emulates 10 end-users and keeps sending HTTP requests to the server, and thus 3 hosts emulate totally 30 concurrent end-users. For the *heavy-traffic* condition, all the settings are the same as the light-traffic condition, but each host emulates 100 end-users and thus 300 concurrent end-users in total.

In the experiment, the end-user state only transits between the *User Communicating State* and the *User Inactive State*, which can be modeled as a 2-state Markov process. Moreover, we suppose that the real-time end-user state can always be correctly identified by the User Model subsystem. When in the User Communicating State, we assume that each end-user makes sequential HTTP requests following a homogeneous Poisson process with a rate of 7 requests per minute. After sending 10 HTTP requests during the User Communicating State, each end-user will automatically transit to the User Inactive State. During the User Inactive State interval, the end-user stops generating the HTTP requests to the Web server until the User Communicating State resumes. In the experiment, the HTTP request generator periodically alternates between the two end-user states and repeats 3 cycles, which means each end-user experiences 3 User Communicating States and 3 User Inactive States. Thus each end-user sends a total of 30 HTTP requests during the experiment. Since the time interval of the User Inactive State varies irregularly and only depends on practical occasions, we simply set this value to 30 s in the experiment. To collect more accurate experimental data, we repeated the experiment 3 times with random starting order of the 3 hosts. Then we average out the observations from all the end-users.

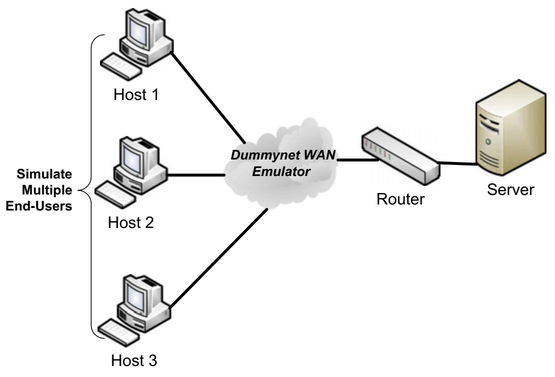
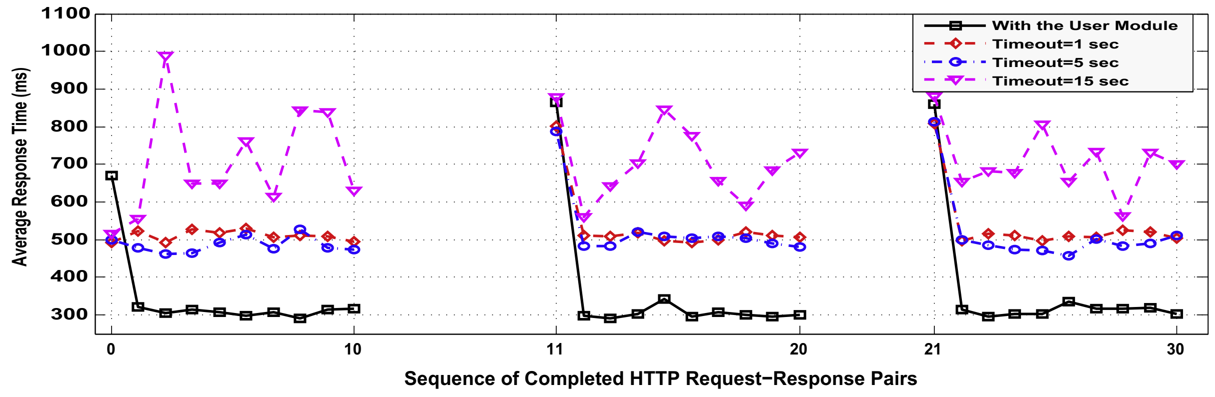


Fig. 6. Network topology in the experiment.

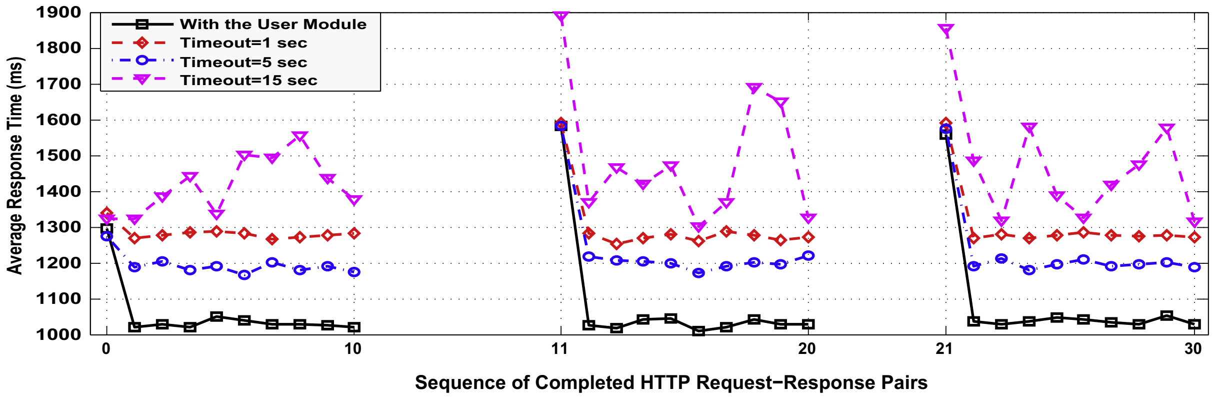
5.5. Experiment results

Based on the above-described experimental setup, we study the network performance and contrast the results with the case without the User Module. We select the Web page response time as the performance metric to evaluate the end-user perceived latency. The Web page response time is the time interval that starts when the end-user sends the Web request and ends when the end-user receives the last object of that Web page. Besides the Web page response time, we also study the Web traffic statistics and the server aggregate throughput to assess the influence of adding the User Module. The collected Web traffic statistics in this experiment includes the number of the successfully delivered Web pages, and the number of HTTP requests transferred between the server and the clients. The Web server aggregate throughput is the sum of the server generated data rates that are successfully provided to all the clients.

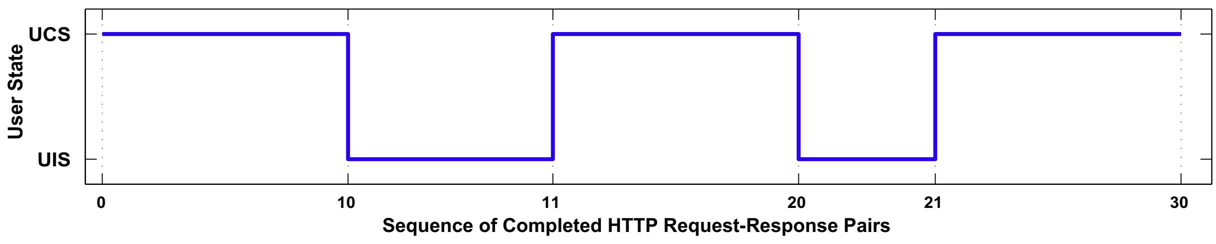
We first investigate the case under the heavy-traffic condition and experiment with two groups of Web pages, which are similar to the SPECweb benchmark [40]. In the first group, the mean size of the generated dynamic Web pages is smaller than 5 KB, and in the second group the mean size is larger than 50 KB. Fig. 7(a) shows the average Web page response time of the small size group, Fig. 7(b) shows that of the large size group, and Fig. 7(c) describes the end-user state transitions during the experiment. From Fig. 7(a) and (b), we see that the User Module case can significantly shorten the average Web page response time of both groups. In contrast to the results where the fixed timeout are used (1, 5 and 15 s), the User Module case can save at least 200 ms on average. It roughly equals to the back-and-forth time on the wire, namely one round trip packet-propagation delay. This is because the User Module can actively extend the lifecycle of the HTTP connections when the end-user stays in the User Communicating State, and thus avoid multiple unnecessary re-establishments of the new HTTP connections. Note that in contrast to the 15 s timeout case under the heavy-traffic condition, the User Module case reduces even more average Web page response time. The main reason is that the fast increasing number of end-users results in a large number of concurrent HTTP persistent connections. With the 15 s fixed timeout value, the Web server cannot terminate



(a) Small Web page size group.



(b) Large Web page size group.



(c) End-user state transition.

Fig. 7. Average Web page response time under the heavy-traffic condition.

inactive connections and allocate the scarce server resource to the newly incoming clients in a timely manner. This causes the number of concurrent persistent HTTP connections to easily reach the upper limit of the Apache HTTP Server, which is set to 256 in this experiment. When this happens, new incoming HTTP connections must wait either in the SYN-queue or the ACK-queue of the Apache Web server. Such queuing delay at the server side can easily amount to several seconds and thus greatly influences the Web page response time. From the collected data of the 15 s timeout case, we see that the Web page response time of the late arriving end-users varies from hundreds to thousands of milliseconds, although the early arriving end-users can still attain quite small response time. Thus in the 15 s timeout case, the high and unstable queuing delay experienced by the late arriving end-users lead to the large average Web page response time and significant fluctuations, which are evident in Fig. 7(a) and (b).

Fig. 8 shows the ratio of the total number of HTTP requests sent by the end-users to the total number of the successfully transferred Web pages. For both the small and the large page size groups, the User Module case achieves the smallest ratio, i.e., 1.11 and 1.08, respectively. The 1 s fixed timeout case results in the highest ratio, i.e., 1.95 and 1.97, which indicates that almost two HTTP requests are required to fetch one Web page. This is because once the small timeout occurs, the Web server will send a TCP segment with the FIN bit set to 1 and enter the FIN_WAIT_1

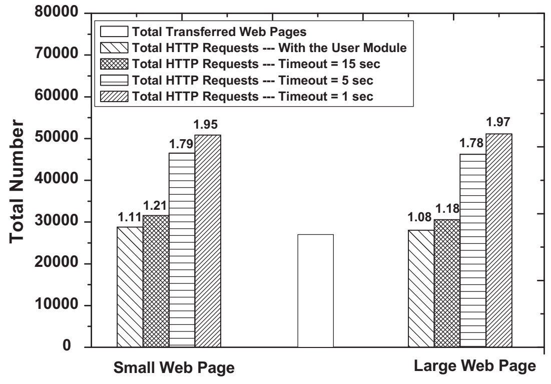


Fig. 8. Ratios of HTTP request number to transferred Web page number.

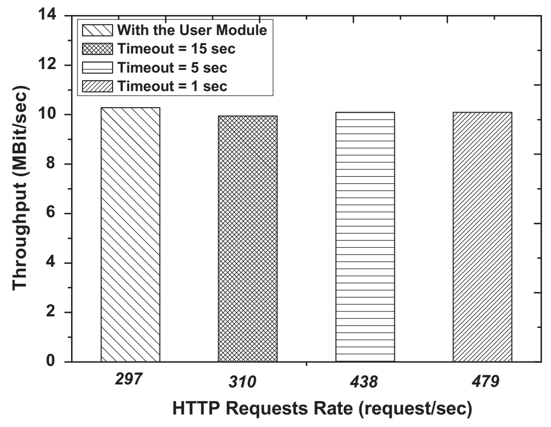


Fig. 9. Throughput of Web server under heavy-traffic.

state. But the Web browser may continue sending new but already invalid HTTP requests through the same connection before it sends the clients' side TCP segment with the FIN bit. Besides the transmission of invalid HTTP requests, the unnecessary re-establishment and closing the HTTP connection also significantly increases the burden on both the Web server and the Internet backbone. Note that in practice most Web browsers usually open multiple concurrent HTTP connections for fetching one Web page and different browsers adopt different mechanisms to reduce the unnecessary retransmission. Accordingly, the absolute value of the ratio may vary case by case, but its relative trend will be the same as shown in Fig. 8.

Under the heavy-traffic condition, the average throughput of the Web server hosting the large Web page group is shown in Fig. 9. We see that introducing the User Module cannot increase the server throughput, since the aggregate throughput does not greatly depend on the timeout value. However, Fig. 9 shows that with the equivalent aggregate throughput, the User Module case achieves the smallest HTTP request rates, and thus also demonstrates the significant reduction of the burden on the server and the Internet backbone.

The experimental results under the light-traffic condition are comparable with the results under the heavy-traffic condition. The User Module case also shortens the average Web page response time, as shown in Fig. 10, and reduces unnecessary traffic burden on the server and Internet. Similar to the heavy-traffic condition, the User Module case reduces almost the equivalent of one round trip packet-propagation delay when compared to the 1 s and 5 s fixed timeout cases. The only difference is that the 15 s timeout case can also achieve quite short average Web page response time under the light-traffic condition. It is because that the number of concurrent active clients never reaches the maximum number of allowable connections in the Web server. Moreover, during the User Communicating State, the time interval between the consecutive HTTP requests is always smaller than 15 s, and thus the HTTP persistent connections will not be terminated by the Web server as frequently as the small timeout value cases.

5.6. Analysis and discussion

Prior studies [46,47] have shown that the Web page response time greatly influences the end-user QoE in Web browsing. With the same QoE rating measurement (called Opinion Scores), the quantitative relationship between the end-user QoE and the Web page response time has been investigated: ITU-T G.1030 [47] demonstrates that the logarithmic relationship fits well, while Shaikh et al. [46] shows that the exponential relationship gives the best correlation result. The two relationships are compared in [21], where a generic exponential relationship between the end-user QoE and the QoS parameters has been suggested. Those results illustrate that the mathematical relationship between the end-user QoE and the Web page response time may vary due to the diversity of the participants in user studies and the network configurations. However, all the derived models verify that the Opinion Scores monotonously increases with the decreasing Web page response time. Therefore, it confirms our initial claims that such User Module application would enhance the end-user QoE through improving the underlying network performance.

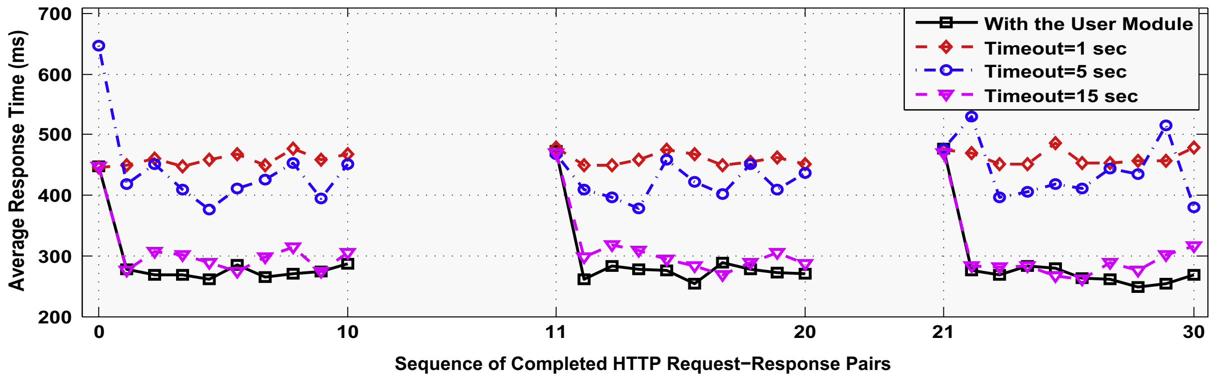
On the other hand, from the experimental results, we see that the current version of HTTP, with either small or large fixed timeout values, is unable to achieve optimal operation in terms of managing its persistent connection mechanism. It is because no end-user state information available for HTTP and even for the whole application layer. Therefore, the built end-user models under the User Module framework is an effective solution to bridge the gap between HTTP and the Web end-users.

6. The User Module application 2: TCP

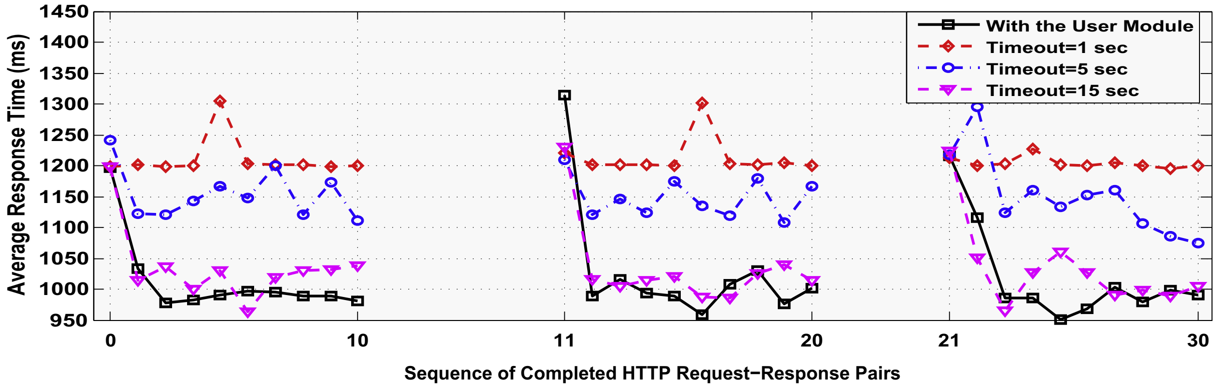
In the second User Module application, the Control subsystem mainly manipulates the advertised window size in TCP protocol based on the derived end-user states. This application aims to enhance the end-user QoE through dynamically allocating the limited bandwidth resource on the transport layer.

6.1. Problem description

As a connection-oriented and reliable Transport Layer communication protocol, Transmission Control Protocol



(a) Small Web page size group.



(b) Large Web page size group.

Fig. 10. Average Web page response time under the light-traffic condition.

(TCP) [48] is an indispensable component of the modern Internet protocol stack. Many Internet services and their applications rely on TCP as their transport carrier, such as File Transferring service (CuteFTP), Web Browsing (Firefox), Electronic Mailing (Microsoft Outlook), and some Live Multimedia Streaming service (QQQTV).

As mentioned earlier, TCP also follows the traditional Internet design principle that assuming two communicating hosts always desire speaking to each other. Hence, an individual TCP stream always intends to maximize its own throughput unless the network congestion or the receiver buffer overflow happens. It has been proved that when multiple TCP streams compete the same *bottleneck link*, the stream with a smaller RTT can always receive a much larger share of that bottleneck link bandwidth than other streams with larger RTT [49]. Therefore, TCP always favors an Internet application with short RTT regardless of the end-user preference and other influential factors. Such TCP property can easily influence, and even impair the end-user QoE, especially when the end-user want to prioritize the Internet application with larger RTT. For instance, an end-user may simultaneously open CuteFTP to download a large file and QQQTV to watch online TV. As a live multimedia streaming application, QQQTV always requires a minimum guaranteed bandwidth, but some CuteFTP connections with small RTT can easily grab most of the available bandwidth at the bandwidth-limited access link, where the last mile bottleneck exists. In order to enable TCP to provide such bandwidth prioritization service for enhancing the end-user QoE, adopting the built end-user models under the proposed User Module architecture is also a natural and effective solution.

6.2. The control subsystem and experimental results

In order to actively allocate network resource at the bandwidth-limited access link, the Control subsystem of the User Module needs to leverage on the flow control mechanism of TCP. The original objective of the TCP flow control mechanism is to obviate the TCP sender overflowing the TCP receiver's local buffer. Different from the well-known TCP congestion control mechanism, the TCP flow control mechanism maintains a variable called *advertised window* at TCP receiver side. The advertised window size is always set to the amount of spare room in the buffer, and is included in each TCP acknowledgement returned to the TCP sender. Thus, the advertised window can actually limit the maximum number of bytes a sender is allowed to transmit before receiving the next acknowledgment from the receiver side.

There has been some prior work on adjusting the advertised window size in [50,51]. Most prior work follows a similar mathematical model describing the relationship between the sending rate V of a TCP connection and its advertised window size:

$$V = \frac{W^{rec}}{RTT}, \tag{1}$$

where W^{rec} is the advertised window size in bits and RTT is the average round trip time of the corresponding TCP connection in seconds. The above mathematical model sufficiently describes the TCP flow control mechanism, although it simplifies the other complicated TCP mechanisms such as the congestion control dynamics [52]. The above model is based on the following assumptions:

- The access link or the last hop link is the bottleneck link of the whole networks, which commonly exists in wired and wireless networks.
- The packet loss probability is small and thus we neglect the effect of the TCP slow start and the time-out mechanism.
- Only long-term bulk-transfer Internet applications are considered, since the short-term small sessions are likely to have completed before they can influence the end-user QoE.

From Eq. (1), we see that adjusting the advertised window size at the receiver side can directly influence the TCP sending rate. Therefore, the Control subsystem can redistribute the limited bandwidth resource at the access link to different Internet applications by manipulating the advertised window size at the TCP receiver side. Furthermore, Eq. (1) also shows that the sending rate V is inversely proportional to the RTT, and the Control subsystem can either employ the TCP timestamp option [53] or the method proposed in [54] to calculate the average RTT. Note that implementing such a control subsystem at the TCP receiver side for adjusting the advertised window size does not require any changes to the existing TCP protocol.

We set up our experiment with the two Internet applications *QQQTV* and *CuteFTP*: the end-user opens QQQTV for watching the online live TV program (ESPN channel), and meanwhile uses CuteFTP to download a zip file (200 MB). To enable the access link to be the bottleneck link in the experiment, we employ NetLimiter [55] at the end-user side to limit the overall incoming throughput to 1.0 Mbps. Since the main objective of such User Module application is to enhance the end-user QoE, we conduct the specific user study to investigate the relationship between the bandwidth consumption of QQQTV and the corresponding end-user QoE. End-users are asked to provide their subjective responses about the QQQTV performance on the Opinion Score scale from 5 to 1. The following grades are used: 5 = Excellent, 4 = Good, 3 = Average, 2 = Poor, 1 = Bad. In total, 4 female and 5 male test users attended the study with an age distribution between 20 and 55 years (mean 32.3 years, median 29.4 years). In terms of watching live streaming multimedia and Internet usage, all participants are rather advanced. The solid line in Fig. 11 depicts the end-user QoE on QQQTV as a function of the allocated bandwidth (from 100 Kbps to 1000 Kbps, error bars representing 95% confidence intervals). The outcome of the user study demonstrates that the Opinion Score is Good when the allocated bandwidth above 400 Kbps, and the Opinion Score drops to almost Poor when the allocated bandwidth below 320 Kbps, i.e., QQQTV's performance deteriorates to an unacceptable level for end-users. For the end-user QoE on CuteFTP, Reichl et al. [56] modeled the QoE on file downloading as a function of file size s and download bandwidth V based on the zero-mean normalization method:

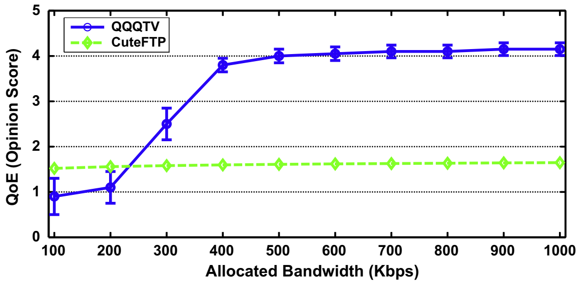


Fig. 11. End-User QoE on QQQTV and CuteFTP as a function of allocated bandwidth.

$$QoE = \frac{0.775}{\sqrt{s}} \ln(V) + 1.268. \tag{2}$$

We adopt the above model to describe the relationship between the bandwidth consumption of CuteFTP and the corresponding end-user QoE. Given the fixed file size 200 MB, the dash line in Fig. 11 depicts the end-user QoE on CuteFTP within the same range of the allocated bandwidth (from 100 Kbps to 1000 Kbps). From Fig. 11, we see that allocating enough bandwidth to QQQTV greatly improves the end-user QoE on QQQTV, while decreasing the bandwidth for downloading large file has limited impairment to the end-user QoE on CuteFTP. Hence, when QQQTV and CuteFTP are running simultaneously and both in the User Communicating State, QQQTV should be given the priority to receive enough limited bandwidth resource. Based on the above experimental results and analysis, the Control subsystem at the end-user side can implement the following control rules:

(1) IF QQQTV is in the User Communicating State or the Unidentified User State and its bandwidth share is *lower* than 320 Kbps, THEN the Control subsystem immediately *reduces* the advertised window size of CuteFTP until QQQTV bandwidth share exceeds 400 Kbps.
(2) IF QQQTV switches back to the User Inactive State or is terminated, THEN the Control subsystem *increases* the advertised window size of CuteFTP to the initial value.
(3) For other possible situations, the Control subsystem takes no action.

The above control rules ensure that when the end-user is watching QQQTV, QQQTV is always given priority to receive enough bandwidth. Meanwhile, CuteFTP can also keep working with the leftover network resource rather than be forcibly paused or closed. When the end-user stops concentrating on QQQTV, the Control subsystem will take back the privilege given to QQQTV and allow the running CuteFTP to fairly compete for the limited bandwidth resource again.

Since we mainly aim to demonstrate the basic design idea on the second application of the User-Context Module, the experiment setup is relatively simple. With more complicated conditions, new Control subsystem and control rules should be re-designed. For example, the Control subsystem at the server side can also be launched to actively allocate the limited network resource among different end-user groups, when a bottleneck link exists at the uplink of the server. Some data mining and machine learning algorithms can also be introduced to automatically generate the control rules based on the end-user interaction data [57].

To achieve fast response and avoid overshoot, we adopt the commonly-used discrete PD (Proportional and Derivative) control algorithm to adjust the advertised window size:

$$\Delta W_k^{rec} = K_p e(k) + K_d(e(k) - e(k-1)), \qquad (3)$$

where

$$e(k) = R - V_k.$$

The controller output ΔW_k^{rec} is used to adjust the advertised window size at the K^{th} sampling time. R is the target bandwidth and V_k is the allocated bandwidth at the K^{th} sampling time. K_p and K_d are the tuning parameters in the PD control algorithm. The sampling interval can be about five RTT of the corresponding TCP connection.

To illustrate the feasibility and effectiveness of the above-described solution, we implement the Control subsystem in Network Simulator-2 (NS-2 Version 2.33) [58]. We fulfill the new function module to calculate the real-time bandwidth consumption for each TCP connection, and add the proposed PD control algorithm at the TCP sink side in NS-2. The simulation parameters are identical to the experiment setting and the end-user state transition on QQQTV is demonstrated in Fig. 12.

Fig. 13 shows that without the User Module, the bandwidth distribution of the bottleneck link (1Mbps in total) when QQQTV and CuteFTP run simultaneously. We see that CuteFTP always captures most of the limited bandwidth resource, i.e., nearly 800 Kbps, because it has relatively smaller average RTT and enough initial advertised window size. Therefore, QQQTV always cannot receive the minimum guaranteed bandwidth, which greatly impairs the end-user QoE on QQQTV. In practice, the end-user has to manually shut down or pause CuteFTP to facilitate normal watching QQQTV.

Fig. 14 illustrates that with the User Module, the bandwidth distribution of the bottleneck link. When the end-user starts watching QQQTV, i.e., QQQTV with the User Communicating State, the User Module can automatically decrease the bandwidth consumption of CuteFTP until QQQTV bandwidth share exceeds 400 Kbps. When the end-user temporarily stops watching QQQTV, i.e., QQQTV switched to the User Inactive State, the User Module detects this change in real-time. The Control subsystem then releases the constraints on the advertised window size of CuteFTP, and takes back the privilege given to QQQTV. Besides, the Control subsystem guarantees that the running Internet applications always utilize the entire bottleneck link capacity.

In short, this exemplary application demonstrates how the User Module empowers end-users to influence bandwidth distribution with the aim of enhancing the end-user QoE. Subsequently, it is necessary to provide a proper and widely applicable approach to evaluate the variance of the end-user QoE.

6.3. End-User QoE with the structured approach

As indicated earlier, end-user QoE is a joint consequence of the technical parameters (traditional QoS parameters), the communication context environment and the characteristics of the network services in use. Since large amount of variables and information need to be considered, Brooks et al. [22] propose a structured approach to end-user QoE with the following clause:

IF ⟨*Communication Situation*⟩;
USING ⟨*Service Prescription*⟩;
WITH ⟨*Technical Parameters*⟩;
THEN ⟨*End-User QoE*⟩.

Such structured approach explicitly combines end-user's usage context variables and technical parameters together to measure end-user QoE, and all the attributes in the bracket have many possible options. For example, ⟨*Communication Situation*⟩ takes into account objective communication context related to end-users. Therefore, the User Communicating State, the User Inactive State and other properly defined end-user states under the User Module framework can be introduced into the parameter set of the ⟨*Communication Situation*⟩ attribute. The ⟨*Service Prescription*⟩ can be Live Multimedia Streaming (QQQTV), File Transferring (CuteFTP) or any other types of Internet services. The ⟨*Technical Parameters*⟩ ranges from the bit rate to the protocol type, and a more complete list is given in [22]. For the ⟨*End-User QoE*⟩, the Opinion Score can still be used to describe end-user satisfaction on the performance of the given Internet service.

With the newly introduced structured approach, we can describe and measure end-user QoE in a more clear and comprehensive way. Accordingly, Fig. 11 has been expressed as follows:

IF ⟨*User Communicating State*⟩;
USING ⟨*QQQTV*⟩ or USING ⟨*CuteFTP*⟩;
WITH ⟨*Bit Rate (from 100 Kbps to 1000 Kbps)*⟩;
THEN ⟨*QoE Results in Fig. 11*⟩.

Note that we take the *Bit Rate* as the main parameter of the attribute ⟨*Technical Parameters*⟩, because it is the key factor to influence the performance of both QQQTV and CuteFTP. Different technical parameters can be considered for other Internet services and applications, e.g., the Web Page Response Time for Web browsing in the first User Module application. Moreover, when the parameter of the ⟨*Communication Situation*⟩ is the User Inactive State,

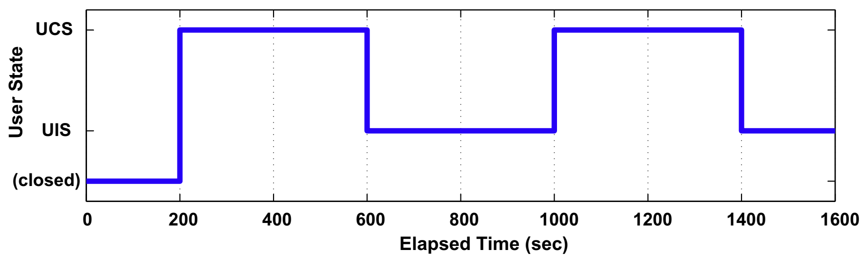


Fig. 12. End-user state transition on QQQTV.

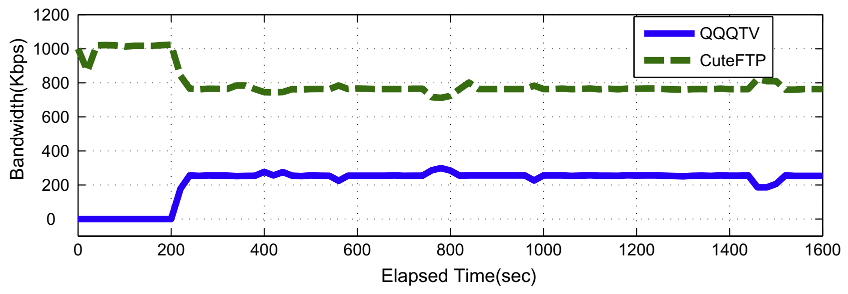


Fig. 13. Bandwidth allocation without the User Module.

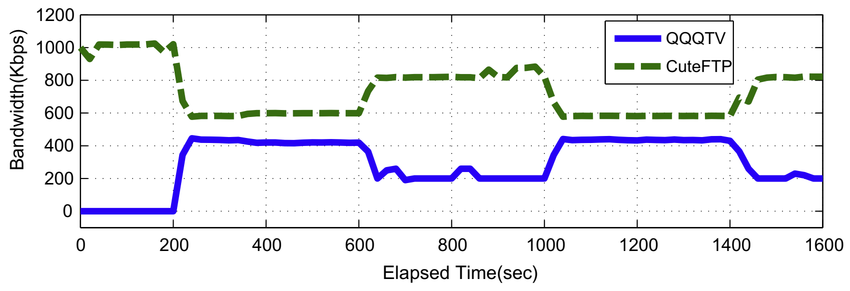


Fig. 14. Bandwidth allocation with the User Module.

the Opinion Score in the ⟨*End-User QoE*⟩ can be simply assigned “0” regardless of the types of Internet services and the variance of technical parameters. The main reason is that the User Inactive State, which has been defined in Section 4, essentially implies no interaction between the end-user’s all three MHP subsystems and the corresponding Internet service, and thus it results little influence on the end-user’s subjective satisfaction.

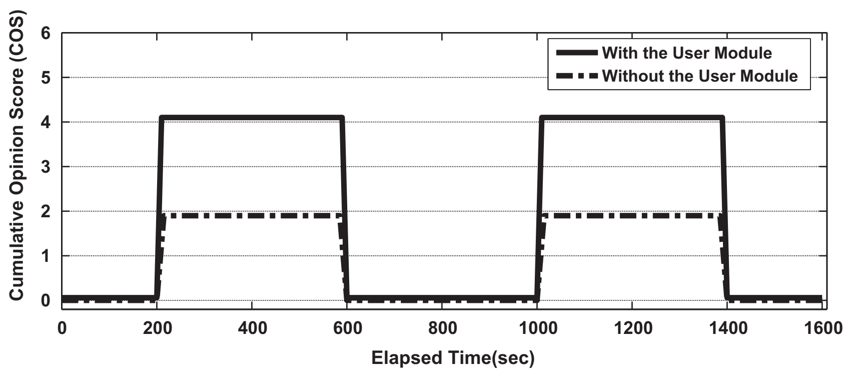
Given the structured approach and the end-user state transition on QQQTV depicted in Fig. 11, we further consider the following two specific scenarios describing the end-user state on CuteFTP:

Scenario A: CuteFTP always with the User Inactive State (End-user is unaware of CuteFTP downloading from the beginning to end).

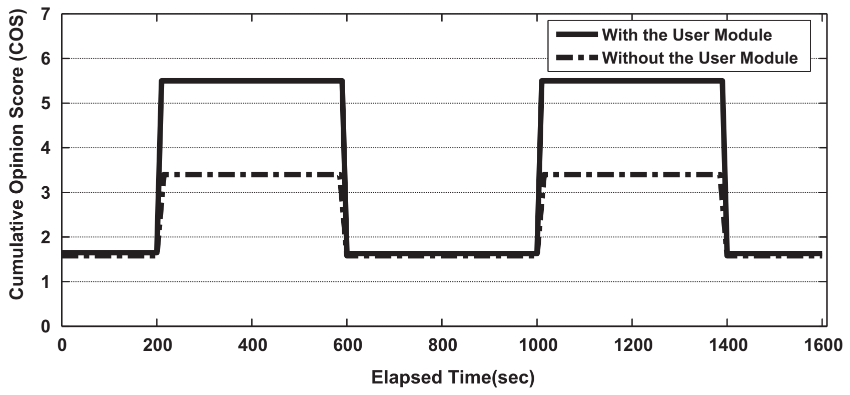
Scenario B: CuteFTP always with the User Communicating State (End-user is aware of CuteFTP downloading even when watching QQQTV).

Based on the bandwidth distribution results in Figs. 13 and 14, we can calculate the Cumulative Opinion Score (COS), i.e., the sum of the Opinion Score on QQQTV and CuteFTP, for the above-described two scenarios, respectively. Fig. 15(a) illustrates the variances of the COS under the Scenario A: since the Opinion Score on CuteFTP keeps zero (CuteFTP always with the User Inactive State), Fig. 15(a) essentially depicts the end-user Opinion Score on QQQTV. Therefore, we clearly see that introducing the User Module effectively prevents the end-user QoE on QQQTV falling down to the Poor level and maintains it at a high satisfaction level during QQQTV with the User Communicating State. Fig. 15(b) demonstrates the variances of the COS under the Scenario B: we see that the User Module can still dramatically increase the COS during QQQTV with the User Communicating State, although the end-user Opinion Score on CuteFTP would be slightly decreased because of the less bandwidth allocated to it. Such result indicates that when the User Module provides high QoE on QQQTV, it does not influence much on the overall QoE of the Interne end-user under the scenario B.

Note that the given scenarios are the two extreme cases depicting the end-user state with CuteFTP. In reality, the end-user is more likely to frequently switch between the



(a) COS under the Scenario A



(b) COS under the Scenario B

Fig. 15. A comparison of Cumulative Opinion Score (COS): With the User Module vs Without the User Module.

two defined user states (and even some unidentified states) caused by human complex internal conscious and unconscious psychological and cognitive factors [59]. With the positive COS results shown in Fig. 15(a) and (b), we can tentatively conclude that the proposed User Module can effectively enhance the end-user QoE in practice with the built Control subsystem. Moreover, the structured approach to end-user QoE efficiently specifies the key parameters and is applicable to a wide range of the User Module applications.

7. Conclusion

In this paper, we rethink the communication network design issues by introducing the AmI concept and end-user factor into the Internet protocol stack. We thus propose the User Module framework: the User Interface subsystem is built to fully sense Internet end-users, the User Model subsystem employs cognitive and HCI knowledge to model and recognize end-user states, and the Control subsystem directly interacts with the network protocols through tuning their critical parameters. The proposed framework are validated by the two User Module applications, which also partly demonstrate the User Module's operations, practices and impacts. The Internet experiments with the evaluation results confirm our initial claims that the proposed User Module can effectively enhance the end-user QoE and improve the underlying network performance.

These contributions lay a solid foundation for future research:

- With the aim of enhancing the Internet as a user-centric AmI communication system, the User Module can explore its interactions with many other critical parameters in all existing protocols and layers, such as the session-related variables in Real Time Streaming Protocol (RTSP) on Application Layer or the security-related variables in Internet Protocol Security (IPsec) on Network Layer.
- Based on the latest progress on the fields of pervasive sensing technology and the cognitive psychology, the User Interface and User Model subsystems can be further developed and upgraded. New significant context information of the end-user can be collected and accordingly explicit end-user states can be defined.
- On the server side, the group user model could help enhance server performance and facilitate batch processing. Designing the group user model and how it can influence overall system performance require further study.

We also hope this work could open up a new realm for innovations on next generation Internet architecture with the *clean-slate* design approach [6]. For example, some standard control interfaces for tuning the adjustable and critical network parameters can be explicitly defined on future Internet protocols or Internet services [60]. Such design would enable the User Module to provide a general solution for the Internet to utilization of the end-user's key context information.

In summary, all these efforts go towards supporting a user-centric, context-aware and highly adaptive communication network.

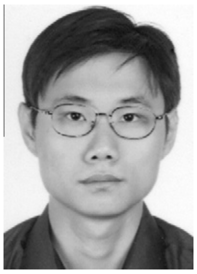
Acknowledgement

This work is partially supported by project grant NRF2007IDMIDM002–069 on "Life Spaces" from the IDM Project Office, Media Development Authority of Singapore.

References

[1] H. Zimmermann, OSI reference model–The ISO model of architecture for open systems interconnection, IEEE Trans. Commun. 28 (1980) 425–432.
[2] J.F. Kurose, K.W. Ross, Computer Networking: A Top-Down Approach, Pearson Education, Inc., 2008.
[3] D.D. Clark, J. Wroclawski, K.R. Sollins, R. Braden, Tussle in cyberspace: defining tomorrow's internet, IEEE/ACM Trans. Netw. 13 (2005) 462–475.
[4] J.H. Saltzer, D.P. Reed, D.D. Clark, End-to-end arguments in system design, ACM Trans. Comput. Syst. 2 (1984) 277–288.
[5] D.D. Clark, The design philosophy of the DARPA internet protocols, SIGCOMM Comput. Commun. Rev. 25 (1995) 102–111.
[6] S. Paul, J. Pan, R. Jain, Architectures for the future networks and the next generation internet: a survey, Comput. Commun. 34 (2011) 2–42.
[7] J. Bravo, L. Fuentes, L.I. Diego, Theme issue: ubiquitous computing and ambient intelligence, Personal and Ubiquitous Computing, 2011
[8] W. Weber, J.M. Rabaey, E. Aarts, Ambient Intelligence, McGraw-Hill Inc., 2001.
[9] MIT, 2004, MIT project oxygen for pervasive human-centered computing (Online), Available: <http://oxygen.lcs.mit.edu/overview.html>.
[10] Gatech, 2010, Georgia Techs Aware Home (Online), Available: <http://awarehome.imtc.gatech.edu/>.
[11] IBM, 2003, Blue Space (Online), Available: <http://www.research.ibm.com/bluespace/index.html>.
[12] R. Lachman, J.L. Lachman, E.C. Butterfield, Cognitive Psychology and Information Processing: An Introduction, Lawrence Erlbaum Associates, 1979.
[13] K.S. Card, P. Thomas, N. Allen, The Psychology of Human Computer Interaction, L. Erlbaum Associates Inc., 1983.
[14] D.E. Meyer, D.E. Kieras, A computational theory of executive cognitive processes and multiple-task performance, Psychol. Rev. 104 (1997) 3–65.
[15] J.L. McClelland, On the time relations of mental processes: a framework for analyzing processes in cascade, Psychol. Rev. 88 (1979) 375–407.
[16] R.W. Proctor, L. Vu Kim-phuong, Handbook of Human Factors in Web Design, Erlbaum, Mahwah, NJ, 2004.
[17] D.S. Dylan, Foundations of Augmented Cognition, Erlbaum, Mahwah, NJ, 2005.
[18] R. Jain, Quality of experience, IEEE Multimedia 11 (2004) 95–96.
[19] Nokia, Quality of Experience (QoE) of mobile services: Can it be measured and improved, White Paper, Finland, 2004.
[20] ITU-T, 2008. Rec. P. 10/G. 100, Amendment 2: New definitions for inclusion in recommendation ITU-T P.10/G.100.
[21] M. Fielder, T. Hossfeld, P. Tran-Gia, A generic quantitative relationship between quality of experience and quality of service, IEEE Netw. 24 (2010) 36–41.
[22] P. Brooks, B. Hestnes, User measures of quality of experience: why being objective and quantitative is important, IEEE Netw. 24 (2010) 8–13.
[23] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufman, San Francisco, 2005.
[24] T. Gu, K.P. Hung, D.Q. Zhang, An ontology-based context model in intelligent environments, in: Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, 2004, San Diego, USA.
[25] V. Kawadia, P.R. Kumar, A cautionary perspective on cross-layer design, IEEE Wireless Commun. 12 (2005) 3–11.
[26] S. Vineet, M. Motani, Cross-layer design: a survey and the road ahead, IEEE Commun. Mag. 43 (2005) 112–119.
[27] R.J. Anderson, M. Matessa, C. Lebiere, ACT-R: a theory of higher level cognition and its relation to visual attention, Hum.-Comput. Interact. 12 (1997) 439–462.
[28] A. Howes, M. Young, Richard, The role of cognitive architecture in modeling the user: Soar's learning mechanism, Hum.-Comput. Interact. 12 (1997) 311–343.
[29] N. Allen, Unified Theories of Cognition, Harvard University Press, Cambridge, MA, 1990.
[30] G. Bradski, A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, O'Relly Media, Inc., 2008.
[31] Z. Ramdane-Cherif, A. Nait-Ali, An adaptive algorithm for eye-gaze-tracking-device calibration, IEEE Trans. Instrum. Meas. 57 (2008) 716–723.
[32] M. Betke, J. Gips, P. Fleming, The camera mouse: visual tracking of body features to provide computer access for people with severe disabilities, IEEE Trans. Neural Syst. Rehabil. Eng. 10 (2002) 1–10.
[33] F. Risso, L. Degioanni, An architecture for high performance network analysis, in: Proceedings of IEEE ISCC, 2001, HAMMAMET, Tunisia.
[34] L. Hui, D. Houshang, B. Pat, L. Jing, Survey of wireless indoor positioning techniques and systems, IEEE Trans. Syst, Man Cybernetics – Part C: Appl. Rev. 37 (2007) 1067–1080.
[35] R. Fielding, J. Gettys, J.C. Mogul, et al., IETF RFC 2616 (1999).
[36] T. Faber, J. Touch, W. Yue, The TIME-WAIT state in TCP and its effect on busy servers, in: Proceedings of IEEE INFOCOM, 1999, New York, NY, USA.
[37] P. Barford, M. Crovella, A performance evaluation of hyper text transfer protocols, ACM SIGMETRICS Perform. Eval. Rev. 27 (1999) 188–197.
[38] J.C. Mogul, The case for persistent-connection HTTP, in: Proceedings of ACM SIGCOMM, 1995, Cambridge, MA, USA.
[39] A. Sugiki, K. Kono, H. Iwasaki, Tuning mechanisms for two major parameters of Apache web servers, Softw. Pract. Exper. 38 (2008) 1215–1240.
[40] SPECweb, 2005, Benchmark Design Document (Online), Available: <http://www.spec.org/web2005>.
[41] P. Barford, M. Crovella, Generating representative Web workloads for network and server performance evaluation, in: Proceedings of ACM SIGMETRICS, 1998, Madison, Wisconsin, USA.
[42] W3C, 2004, Libwww: The W3C protocol library (Online), Available: <http://www.w3.org/Library>.
[43] L. Rizzo, 2010, Dummynet home page (Online), Available: <http://info.iet.unipi.it/luigi/dummynet/>.
[44] S. Elnikety, E. Nahum, J. Tracey, W. Zwaenepoel, A method for transparent admission control and request scheduling in e-commerce web sites, in: Proceedings of ACM WWW, 2004, New York, NY, USA.
[45] B. Schroeder, M. Harchol-Balter, Web servers under overload: how scheduling can help, ACM Trans. Internet Technol. 6 (2006) 20–52.
[46] J. Shaikh, M. Fiedler, D. Collange, Quality of experience from user and network perspectives, Ann. Telecommun. 65 (2010) 47–57.
[47] ITU-T, 2005. Rec. G. 1030: estimating end-to-end performance in IP networks for data applications.
[48] J.B. Postel, Transmission control protocol, information science institute, RFC 793 (1981).
[49] T.V. Laskshman et al., The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM Trans. Networking 5 (1997) 336–350.
[50] P. Mehra, C.D. Vleeschouwer, A. Zakhor, Receiver-driven bandwidth sharing for TCP and its applications to video streaming, IEEE/ACM Trans. Multimedia 7 (2005) 740–752.
[51] N.T. Spring, M. Chesire, M. Berryman, et al., Receiver based management of low bandwidth access links, in: Proceedings of IEEE INFOCOM, 2000, Tel-Aviv, Israel.
[52] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling TCP Reno performance: a simple model and its empirical validation, IEEE/ACM Trans. Networking 8 (2000) 133–145.
[53] V. Jacobson, R. Braden, B.D., TCP Extensions for High Performance, Information Science Institute, RFC 1323, 1992.
[54] Berzosa et al., Receiver-based RTT measurement in TCP, U.S. Patent 7330426 B2 (2008).
[55] L. Software, 2010, Internet traffic control and monitoring tool – Netlimiter (Online), Available: <http://www.netlimiter.com/>.
[56] P. Reichl, S. Egger, R. Schatz, A. D'Alconzo, The logarithmic nature of QoE and the role of the Weber–Fechner law in QoE assessment, in: Proceedings of IEEE ICC, 2010, Cape Town, South Africa.
[57] Y. Lu, M. Motani, W.C. Wong, Intelligent network design: user layer architecture and its application, in: Proceedings of IEEE SMC, 2010, Istanbul, Turkey.

[58] UCB/LBNL/VINT, 2005, The Network Simulator Version 2, ns-2 (Online), Available: <http://www.isi.edu/nsnam/ns>.
[59] A. Sears, J.A. Jacko, The Human–Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, Second Edition (Human Factors and Ergonomics), Lawrence Erlbaum Associates, 2002.
[60] R. Dutta et al., The SILO architecture for services integration, control, and optimization for the future internet, in: Proceedings of IEEE ICC, 2007, Glasgow, Scotland.



Yu Lu received the B.E. degree and the M.E. degree in electrical engineering from Beijing University of Aeronautics and Astronautics, China. Since 2008, he has been working towards the Ph.D. degree at National University of Singapore. His research interests include Internet architecture and network protocol design, Web technology, ubiquitous computing, and applications of control theory. Mr. Lu has received the NGS fellowship from NUS graduate school for integrative sciences and engineering.

Mehul Motani received the B.S. degree from Cooper Union, New York, NY, the M.S. degree from Syracuse University, Syracuse, NY, and the Ph.D. degree from Cornell University, Ithaca, NY, all in Electrical and Computer Engineering.

He is currently an Associate Professor with the Electrical and Computer Engineering Department at the National University of Singapore (NUS). He has held a Visiting Fellow appointment with Princeton University, Princeton, NJ. Previously, he was a Research Scientist at the Institute for Infocomm Research in Singapore for three years and a Systems Engineer at Lockheed Martin in Syracuse, NY for over four years. His research interests are in the area of wireless networks. Recently he has been working on research problems which sit at the boundary of information theory, networking, and communications, with applications to sustainable development and societal networks.

Dr. Motani has received the Intel Foundation Fellowship for his Ph.D. research, the NUS Faculty of Engineering Innovative Teaching Award, and placement on the NUS Faculty of Engineering Teaching Honors List. He has served on the organizing committees of ISIT, WiNC and ICCS, and the technical program committees of MobiCom, Infocom, ICNP, SECON, and several other conferences. He participates actively in IEEE and ACM and has served as the secretary of the IEEE Information Theory Society Board of Governors. He is currently an Associate Editor for the IEEE Transactions on Information Theory and an Editor for the IEEE Transactions on Communications.

Wai-Choong Wong received the B.Sc. (first class honors) and Ph.D. degrees in electronic and electrical engineering from Loughborough University, Loughborough, U.K. He is a Professor in the Department of Electrical and Computer Engineering, National University of Singapore (NUS). He is currently Deputy Director (Strategic Development) at the Interactive and Digital Media Institute (IDMI) in NUS. He was previously Executive Director of the Institute for Infocomm Research (I2R) from November 2002 to November 2006. Since joining NUS in 1983, he served in various positions in department, faculty and university levels, including Head of the Department of Electrical and Computer Engineering from January 2008 to October 2009, Director of the NUS Computer Centre from July 2000 to November 2002, and Director of the Centre for Instructional Technology from January 1998 to June 2000. Prior to joining NUS in 1983, he was a Member of Technical Staff at AT&T Bell Laboratories, Crawford Hill Lab, from 1980 to 1983. His research interests include wireless networks and systems, multimedia networks, and source-matched transmission techniques with over 200 publications and four patents in these areas. He is coauthor of the book Source-Matched Mobile Communication (IEEE Press, 1995).

Prof. Wong received the IEEE Marconi Premium Award in 1989, the NUS Teaching Award in 1989, the IEEE Millennium Award in 2000, the e-innovator Awards in 2000, and the Open Category, and Best Paper Award at the IEEE International Conference on Multimedia and Expo (ICME) in 2006.