Available online at www.sciencedirect.com

SCIENCE DIRECT®

J. Parallel Distrib. Comput. 66 (2006) 887–906

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Fractal: A mobile code-based framework for dynamic application protocol adaptation

H. Lufei, W. Shi*

Department of Computer Science, Wayne State University, 420 State Hall, 5143 Cass Ave, Detroit, MI 48202, USA

Available online 11 May 2006

Abstract

The rapid growth of heterogeneous devices and diverse networks in our daily life, makes it is very difficult, if not impossible, to build a one-size-fits-all application or protocol, which can run well in such a dynamic environment. Adaptation has been considered as a general approach to address the mismatch problem between clients and servers; however, we envision that the missing part, which is also a big challenge, is how to inject and deploy adaptation functionality into the environment. In this paper we propose a novel application level protocol adaptation framework, Fractal, which uses the mobile code technology for protocol adaptation and leverages existing content distribution networks (CDN) for protocol adaptors (mobile codes) deployment. To the best of our knowledge, Fractal is the first application level protocol adaptation framework that considers the real deployment problem using mobile code and CDN. To evaluate the proposed framework, we have implemented two case studies: *an adaptive message encryption protocol* and *an adaptive communication optimization protocol*. In the adaptive message encryption protocol, Fractal always chooses a proper encryption algorithm according to different application requirements and device characteristics. And the adaptive communication optimization protocol is capable of dynamically selecting the best one from four communication protocols, including `Direct sending`, `Gzip`, `Bitmap`, and `Vary-sized blocking`, for different hardware and network configurations. In comparison with other adaptation approaches, evaluation results show the proposed adaptive approach performs very well on both the client side and server side. For some clients, the total communication overhead reduces 41% compared with no protocol adaptation mechanism, and 14% compared with the static protocol adaptation approach.

Keywords: Application protocol adaptation; Mobile code; Content distribution networks; Communication optimization; Adaptation

1. Introduction

With the development of computer and communication technologies, more and more heterogeneous devices, like desktops, laptops, PocketPCs, and cellular phones are connected to the Internet using diverse networks, like Ethernet, Wi-Fi, Bluetooth, 3G/4G wireless technology. On one hand, different technologies have different characteristics. On the other hand, a heterogeneous environment makes it possible to dynamically change between different devices and network environments. For instance, a person uses a laptop with a cable modem at home, a cell phone with 3G/4G or Bluetooth on the way to the office, a desktop with Ethernet LAN in the office and a PDA with Wi-Fi in the meeting room. Diverse network connections and heterogeneous devices demand the adaptation functionality in a distributed fashion because no one-size-fits-all single function or protocol can perform well over all these networks and devices.

It is difficult, if not impossible, to build a one-size-fit-all application or protocol which can run well in the dynamic environment. Adaptation has been considered as a general approach to address the mismatch problem between clients and servers [17,27,39,55]. From the perspective of adaptation locations, some of them propose the in-network adaptation, such as CANS [17], Active Names [55], Odyssey [39], and Rover [27], which focus on how to do the adaptation step by step across an overlay path. Although the functionalities are well designed, they have not considered the deployment of chosen components (drivers in CANS [17]) across multiple nodes in

* Corresponding author. Fax: 313 577 6868.

E-mail addresses: hlufei@wayne.edu (H. Lufei), weisong@wayne.edu (W. Shi).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.03.004

the path. This is an obstacle for the wide acceptance of these approaches. Other proposals try to perform the end-to-end adaptation, like the static content-based adaptation [36,41], which does not take the mobility of users and dynamically changing environment into consideration. From the network OSI model's point of view, some of them work in the network layer [43], which adapts the TCP/IP protocol dynamically according to the changing situations on both ends. Although the results are promising, it is not able to handle the application level protocol adaptation which makes more sense for many overlay distributed applications, e.g., streaming multicast on the Internet.

In this paper we propose Fractal, a dynamic application level protocol adaptation approach, which uses the mobile code technology for protocol adaptation and leverages existing content distribution networks (CDN) for protocol adaptors (PADs) (mobile codes) deployment. The idea of protocol adaptation is based on the assumption that an application protocol is composed of a series of components, also called PADs in the Fractal framework. When a protocol needs to be adapted, the application simply needs to add or remove some PADs into or from it. Before a mobile client starts an application session with the application server, it uses the proposed interactive negotiation protocol to negotiate with the adaptation proxy deployed close to the application server. The negotiation manager inside the adaptation proxy uses the proposed adaptation path search algorithm to find one or more appropriate PADs that should be used in the following communication between the client and the application server. Metadata about these PADs will be sent to the client by the adaptation proxy. The client is then able to retrieve the PADs, which are packaged into mobile code modules, from the CDN and starts the new protocol. Although a large amount of research on mobile code and CDN has been done, few studies have combined the advantage of both of them for the protocol adaptation purpose. Based on the proposed framework, we have designed and implemented two case studies: an adaptive message encryption protocol and an adaptive communication optimization protocol. Specifically our contributions of this paper include:

(1) *Proposing a general framework for dynamic application level protocol adaptation*: To the best of our knowledge, Fractal is the first approach on utilization of mobile code in application level protocol adaptation. With the appearance of more and more application level protocols, such as SOAP [56], LDAP [31], and Plugins, holding all the protocol implementations locally is too expensive for some network-enabled mobile devices. Dynamically retrieving the necessary protocol module in an on-demand manner is applicable for mobile hosts.
(2) *Dynamically adapting at the application protocol level*: Most of proposed protocol adaptation methods [4,25,40, 43,50] lie in the network layer. Such systems can cope with localized changes in network conditions but cannot adapt to variations above the network layer. Moreover, their transparency hinders composability of multiple adaptations. Fractal works in the application level so it has the overall system level view to overcome this shortcoming and can maximally adapt application level protocols which have no way to be implemented in the network layer.
(3) *Leveraging CDN edgeservers for protocol adaptor delivery*: CDN has already been widely deployed on the Internet to deliver Web content. Fractal extends the utilization of the content distribution network into the field of protocol adaptation. Considering PAD as a Web object, many algorithms and approaches designed for content distribution on CDN can be seamlessly transplanted to the mobile code distribution scenario. Leveraging existing CDN platforms to deliver PADs for application servers makes our approach more compatible, applicable, and extensible.
(4) *Designing and implementing an adaptive message encryption protocol in the context of the Fractal framework*: Message encryption for secure communication is an important issue in building distributed applications. Many symmetric or asymmetric encryption algorithms have been proposed. Given their dissimilar computing characteristics and the heterogeneity of devices, we argue that it is impossible to ask all applications running on top of diverse devices to choose one encryption algorithm. The only way to accelerate the deployment of encryption algorithms is providing the flexibility of choosing multiple diverse algorithms. We have implemented such an adaptive encryption protocol using Fractal, which dynamically chooses a proper encryption algorithm based on application-specific requirements and device configurations.
(5) *Proposing and implementing an adaptive communication optimization protocol in the context of the Fractal framework*: Many communication optimization techniques are proposed in different contexts. In our previous work [32], we systematically evaluated four algorithms and found that no single algorithm outperformed others in all cases. Different approaches have different performance in terms of different network types, document types, and device configurations. Considering these communication optimization techniques as application level protocols, we implement Fractal in a real system that dynamically chooses different communication optimization protocols and generates the application content for different client devices and network connections. Results show that using framework greatly improves both the client side and server side performance, e.g., the system capacity, client total delay, and bandwidth requirements.

The rest of the paper is organized as follows. After a brief introduction of background in Section 2, Fractal design is depicted in Section 3. Section 4 evaluates the system capacity of the Fractal framework. After that, two case studies about message encryption and communication optimization are presented and evaluated in Sections 5 and 6. Finally, related work and conclusions are listed in Sections 7 and 8 respectively.

2. Background

Our work is inspired by three types of previous work: mobile code [18,24], content distribution network [2,30], and protocol adaptation [34,43,49]. In this section, we explain the general background of each related research field.

2.1. Mobile code

Mobile code [24] is defined as the data that can be executed as a program. The code can be pre-compiled for immediate execution on the recipient's processor, compiled upon receipt for subsequent execution or interpreted. The mobile code system has been used to build a distributed processing environment that is flexible in the communication abstractions it provides to applications and to enhance existing distributed applications. For the benefit of mobile code [18], a major asset provided by code mobility is that it enables service customization. The ability to request the remote execution of code helps increase application server flexibility without permanently affecting the size or complexity of the server. In Fractal we implement each protocol adaptor as a mobile code module, which is sent and executed remotely on the client side to build a new protocol allowing the client to talk with the application server.

2.2. Content distribution network

CDN [30] is an intermediate layer of infrastructure between origin servers and clients. CDN can achieve scalable content delivery by distributing load among its edgeservers, by serving client requests from edgeservers that are close to requests, and by bypassing congested network paths. Currently CDNs are only used to deliver Web-based content. In Fractal framework, CDN is used to deliver PAD. If we consider the PAD as a Web-based object, most of the current techniques in CDN can be leveraged to the delivery of PAD. Fractal framework extends the utilization of CDNs from traditional Web-based content to Web-based objects like mobile code and mobile agent.

2.3. Protocol adaptation

Changing protocols to adapt link condition and network environment is not the new idea, e.g., Reno and Vegas congestion control in TCP/IP protocol [21] is a kind of adaptation. More sophisticated protocol adaptation approaches, such as STP proposed in [43], but most of them are in the network layer which makes them hard to have a general view of the whole system status. The problem of adapting to a changing network environment is further complicated because changes in network conditions are usually transparent to higher layers of the protocol stack. When higher layers, e.g., application layer, are aware of network variation, protocol adaptation can be done more adaptively and intelligently. Based on these observations, Fractal works entirely in the application layer to adapt the application protocol according to heterogeneous client environments.

3. Fractal design—an application protocol adaptation framework

In this section we present the design of Fractal, an application protocol adaptation framework using mobile code and content distribution network edgeservers. After an overview of the Fractal framework, we in turn cover the adaptation proxy, the interactive negotiation protocol, the application protocol adaptation approach, and finally, the mobile code security mechanism.

3.1. System overview

Fractal works entirely at the application level and has no specific requirements about underlying network topologies, connection media types, network protocols, and client hardware configurations. As an general adaptation framework, it focuses on the protocol adaptation method which uses PADs to describe the application protocol structure and distributes the PADs to the client by CDNs for protocol the adaptation purpose. Fractal consists of five components: *application servers*, *adaptation proxies*, *CDN edgeservers*, *PADs*, and *client hosts* (e.g., desktop, laptop, PocketPC, and so on), as shown in Fig. 1. The application server is the application service provider. In order to provide the functionality to heterogeneous clients in diverse environments, the application server usually communicates with clients through different application protocols. For the same application, different content (required) generated by different protocols is called *adaptive content*. For example, the content in a Web page can be transmitted or adapted using either HTTP protocol or HTTPS protocol, which is a more secure mechanism. The HTTP and HTTPS content are called *adaptive content*, as defined earlier. In Fractal, *adaptive content* can be generated either reactively or proactively. The former is suitable for the case in which content keeps changing, e.g., a stock price web site. In this scenario, memory or hard disk space requirements are small, but the price of computing the dynamic *adaptive content* maybe high. On the contrary, the latter, where *adaptive content* is precalculated in advance and saved in memory or disk consumes less CPU and has large memory or disk space requirements. The results in Section 6 show the difference between these two approaches in terms of total time.

Although the application server can talk in many languages, i.e., protocols, the client may not have the necessary protocol to talk with the application server. To help the client talk with the application server, in Fractal we propose the notion of PAD, which is a protocol adaptor implemented in a mobile code module and deployed across the CDN edgeservers. By downloading and deploying one or more PADs, the client is then capable of starting communication with the application server using required protocols. On the server side, we assume the application server has already deployed all PADs in advance. An important issue for the client is which PADs should be used and where to find them. In the Fractal framework, close to the application server, an adaptation proxy is

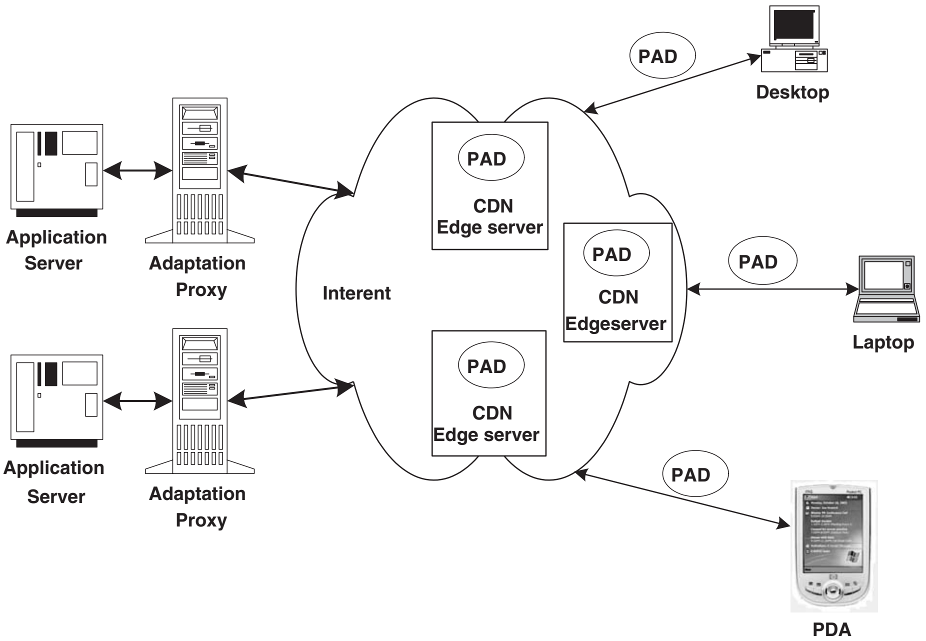

Fig. 1. Architecture of application protocol adaptation using mobile code.

set up to handle the issues about PAD negotiations. Before the initialization of communication between the client and the application server, the client has to negotiate with the adaptation proxy to find proper PADs. The client will be asked to provide some metadata about his environments, such as computing ability, memory space, and network configurations to the adaptation proxy. Having these metadata, the adaptation proxy will generate the metadata of the proper PADs for the client and send the metadata of PADs back to the client. Inside these metadata is enough information for the client to download the PADs from the closest edgeserver of CDNs with which the application server is associated. We will give more details about how the adaptation proxy works in the next section. Fractal leverages the wide deployment of CDNs to distribute the PADs for application servers, as illustrated in Fig. 1. We envision that using CDN edgeservers for application server-specific PADs is a natural extension to the well-known Web content delivery. Note that in this paper we focus on the client/server model; however, it is straightforward to support the peer-to-peer model.

3.2. Adaptation proxy

Adaptation proxy plays an important role in the functionalities of the Fractal framework. Usually it is deployed in the same administration domain as the application server and is responsible for negotiation with the client. A general structure of the adaptation proxy is shown in Fig. 2, which includes a *negotiation manager* module and a *distribution manager* module. Each module is running as a daemon on the adaptation proxy. Next we will explain the structure and functionality of each module respectively.

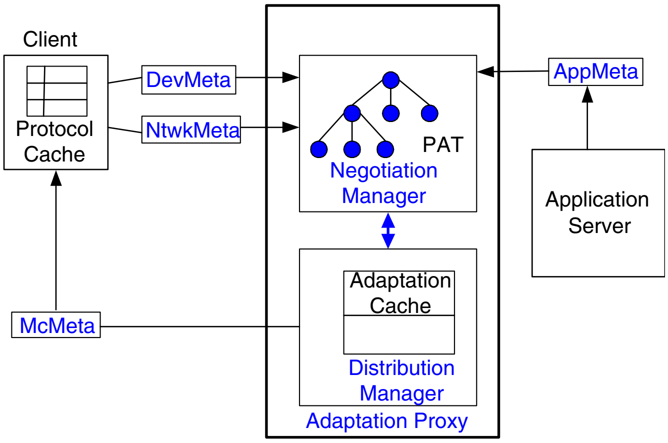

Fig. 2. Structure of the adaptation proxy.

Negotiation manager: As shown in Fig. 2, the negotiation manager is the key in the adaptation proxy which negotiates with the client. Some application level metadata is needed to be transmitted between the adaptation proxy and the application server, and between the adaptation proxy and the client to support the negotiation function. We define these metadata formats in Fig. 3. In the rest of the paper, we will use the acronyms in the parentheses to refer to them. *DevMeta* and *NtwkMeta*, provided by clients, contain the hardware information and the network environment of the client. The application server supplies *PADMeta* to the negotiation manager, who holds the general information of each PAD. *PAD ID* is a unique identification generated by the application server. *PAD overhead* consists of the computing overhead at both the client side and server side, and corresponding traffic overhead, which is hap-

Device Metadata (***DevMeta***) = { Operating system type, CPU type, CPU speed, memory size }

Network Metadata (***NtwkMeta***) = { Network type, Network bandwidth }

PAD Metadata (***PADMeta***) = { PAD ID, PAD size, PAD overhead, Message digest, URL, Parent link, Child link, ... , Child link }

Application Metadata (***AppMeta***) = { Application ID, *PADMeta* 1, ... , *PADMeta* n }

Fig. 3. Definitions of metadata.

pened in the network. *Message digest* is computed using the SHA-1 [13] function and used by clients to verify the integrity of the PAD. *URL* is the link to download the PAD. Note that it is the CDN's responsibility to find the closest edgeserver which holds the PAD, and to redirect the request to that edgeserver. *Parent link* and *Child link* are used to build the protocol adaptation topology in the negotiation manager. *AppMeta* is comprised of *Application ID*, which marks different applications, and some *PADMeta*, which forms a protocol adaptation topology. The application server pushes new *AppMeta* to the negotiation manager when the protocol adaptation topology is first created or changed later. Usually the protocol adaptation topology is represented by a protocol adaptation tree (PAT) structure as shown in Fig. 2 in the upper box located in the negotiation manager in . We will give more details about why a tree is needed and how to build and use the PAT in Section 3.4.1.

When the negotiation manager receives a request from a client, it first checks its adaptation cache, located in the distribution manager. The cache has entries mapping client side information to an array of *PADMeta* that the client needs. Each mapping entry is structured as follows:

$$\{DevMeta, Application\ ID, NtwkMeta\} \Rightarrow$$
$$\{PADMeta\_1, \ldots, PADMeta\_n\}.$$

If the adaptation cache does not have the entry corresponding to the client side metadata, the negotiation manager then will use the algorithm described in Section 3.4.2 to form a new entry and transfer it to the distribution manager.

Distribution manager: The distribution manager is in charge of further processing of these *PADMeta* received from the negotiation manager, updating the adaptation cache, and finally sending *PADMeta* back to the client. When the distribution manager receives the *PADMeta* generated by the negotiation manager, it inserts message digest and URL data into the *PADMeta* and hides the parent and child links since the exposure to the client is unnecessary. After the negotiation procedure, which will be discussed in the following section, the distribution manager will update the adaptation cache so that the negotiation result can be directly retrieved from the cache if the same client configuration occurs later. Finally the distribution manager will handle the network communication details and send these *PADMeta* back to the client. Next we will explain the interactive negotiation protocol.

3.3. Interactive negotiation protocol

In Fractal, an interactive negotiation protocol is proposed for the interactions among these components, as shown in Fig. 4. We assume both the client side and server side understand the protocol definitions. The application server has pre-deployed PADs in the application context and already pushed the *AppMeta* to the adaptation proxy, which has built a PAT inside the negotiation manager. The PADs have been distributed across the CDNs edgeservers.

At the beginning of the negotiation, a client first checks its own protocol cache, which contains some *PADMeta* saved for previous requests. If there is an entry of the protocol cache which matches the current request, the client will directly start the application communication with the application server. If not, the client sends `INIT_REQ`, which contains application request in payload, to the adaptation proxy[1] to initialize the protocol negotiation. Each packet has an *INP header* segment, which is used to maintain the interactive negotiation protocol integrity, and we will omit the details in the *INP header*. The adaptation proxy then sends `INIT_REP` as well as `Cli_META_REQ`, having empty *DevMeta* and *NtwkMeta* to be filled by the client, to acknowledge the request and ask some information about the client. After getting the reply, the client gets the content of *DevMeta* and *NtwkMeta* locally by probing the system using system calls and sends out the `Cli_META_REP`. Based on the `Cli_META_REP`, *PADMeta* is computed and sent back to the client in `PAD_META_REP` by the adaptation proxy. Next, the client updates his protocol cache and sends `PAD_DOWNLOAD_REQ` containing PAD ID to the URL of the PAD. The CDN will automatically choose a close CDN edgeserver and send back the PAD code in `PAD_DOWNLOAD_REP`. If multiple PADs are required, it is not necessary that those PADs downloaded from the same edgeserver. It is up to the CDN to manage the delivery of PADs. After the security check and PAD(s) deployment, the client sends out the `APP_REQ` to the application server. The `APP_REQ` contains the application request as well as the negotiated protocol identifications, which notify the application server to choose the proper PADs to talk with the client. From now on the client and the application server continue the application session using the negotiated protocol. The formats of all message types used in INP are listed on the bottom of Fig. 4.

[1] Note that the client does not have to realize the existence of the adaptation proxy. The application server will automatically redirect the request to its corresponding adaptation proxy.

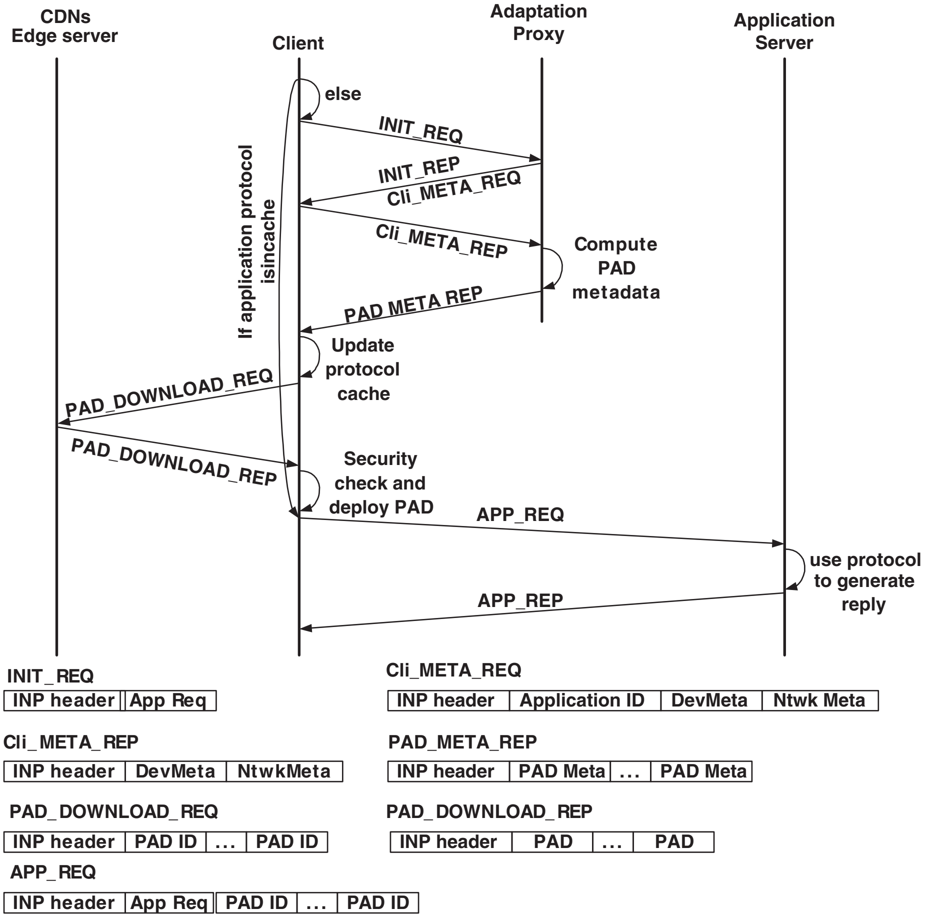
CDNs Edge server
Client
Adaptation Proxy
Application Server
else
If application protocol isincache
INIT_REQ
INIT_REP
Cli_META_REQ
Cli_META_REP
Compute PAD metadata
PAD META REP
Update protocol cache
PAD_DOWNLOAD_REQ
PAD_DOWNLOAD_REP
Security check and deploy PAD
APP_REQ
use protocol to generate reply
APP_REP
INIT_REQ
INP header | App Req
Cli_META_REQ
INP header | Application ID | DevMeta | Ntwk Meta
Cli_META_REP
INP header | DevMeta | NtwkMeta
PAD_META_REP
INP header | PAD Meta | ... | PAD Meta
PAD_DOWNLOAD_REQ
INP header | PAD ID | ... | PAD ID
PAD_DOWNLOAD_REP
INP header | PAD | ... | PAD
APP_REQ
INP header | App Req | PAD ID | ... | PAD ID

Fig. 4. The interactive negotiation protocol.

3.4. Application protocol adaptation

Application protocol adaptation is the major function of Fractal. After introducing the structure and components of Fractal, we will show how the application protocol adaptation works. First, we will explain the protocol adaptation topology, the PAT, which is the main data structure in the procedure of adaptation. Then we will clarify the adaptation path search algorithm.

3.4.1. PAT

Fig. 5 shows an example of the PAT, which is built by the negotiation manager based on *AppMeta* received from the application server. Each node of PAT is a protocol adaptor. The child PAD is an auxiliary component of the parent PAD. In order to run the parent PAD, one and only one of the children PADs must work together with the parent PAD. For example, in Fig. 5, if PAD2 is the FTP protocol, PAD7 is the TCP protocol, and PAD8 is the UDP protocol, the PAD2 can choose either PAD7 or PAD8, but not both. In the real application, it is possible that one PAD is needed by multiple PADs, like TCP protocol is needed by both FTP and HTTP protocols. For the purpose of maintaining the tree structure, we use a symbolic copy of the child PAD if it is required by more than one parent PAD. For instance, in Fig. 5, PAD6 is a symbolic link of PAD7, which is needed by both PAD1 and PAD2. So in order to satisfy an application protocol, a path should be found from the root application to one leaf, e.g., the path composed of PAD2 and PAD7 in the dotted line in Fig. 5. Tree structure makes it flexible enough to extend adaptation protocols by adding new PAD nodes later. For example, if a new PAD, which supports PAD3, is needed later, we just add this new PAD as the first child of PAD3. Adding a new PAD in the middle, instead of the leaf of the tree, can also be done in reasonable time. From the knowledge of data structure and graph theory, we know that the number of possible paths equals the number of leaves in the tree. Next, we propose an adaptation path search algorithm to find the path.

3.4.2. Adaptation path search algorithm

The goal of the adaptation path search algorithm is to find some PADs from PAT to form an adaptation path for a client. Introducing a new protocol into an existing application will inevitably have two effects. First is the traffic overhead, which is either increasing or reducing. Second is the extra computing overhead on both the server side and client side.

Before we choose the proper PADs for a client, the total overhead including traffic and computing overhead of each PAD is the metrics we should quantify. Running each PAD on each

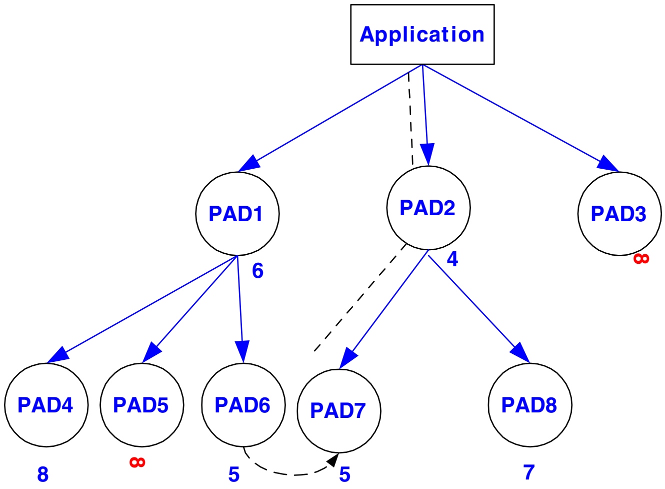

Fig. 5. The protocol adaptation tree.

client configuration and each network environment to get the overhead is not a wise solution. Instead, we use a linear model and a normalized ratio to estimate these overheads. Our linear model comes from the observation that the computing overhead of each PAD is roughly proportional to the processor speed, and the traffic overhead is proportional to the network bandwidth. If the computing overhead of a PAD on one processor speed is known, the computing overhead on another processor can be deducted from the linear ratio of the speed of these two processors. Similarly we can get the traffic overhead of a PAD based on the value of another PAD and the ratio of the bandwidth of two networks. However, this linear model is not so accurate because other parameters of the processor and networks introduce error into the linear model. For example, a scientific computing module is awkward for a no floating instructor processor. A media stream application runs fluently in LAN but not in Dialup. Furthermore, an operating system is also an influential issue that we have to consider besides the linear model. For example, Microsoft DCOM can run on Windows platforms but not Unix environment. In this paper we abstract normalized ratio parameters about three key properties: *processor types*, *operating system*, and *network types* as shown in the normalized ratio matrix in the following context. Note that it is easy to introduce more parameters if necessary, e.g., the screen resolution.

As shown in Eq. (1), each application server maintains the following information. $PAD_{traffic}$ is the traffic overhead of the PAD based on a standard network bandwidth, $Std_{bandwidth}$, 1 Mbp s, and a fixed size of traffic, 1 MB in our implementation. PAD_{size} is the size vector of each PAD. PAD^{client}_{comp} is the computing overhead of PAD on a standard processor speed, Std_{cpu}, 500 MHz Pentium IV in our implementation, on the client side. PAD^{server}_{comp} is the computing overhead of the PAD on the server side, which is supposed to be available in advance. All these metrics can be computed in advance. Later we will compute the estimated overhead of each PAD (PAD_{total}) for a client with specific processor speed and network bandwidth using the linear model plus the normalized ratio matrix. Specifically, we use normalized ratio matrix $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{R}$, as shown in Eq. (2), to measure the performance ratios of n number of PADs on a kinds of processor types, on b number of operation system types, and in r types of network environments. For example,

$$\begin{array}{cc} & \begin{array}{cc} WinCE & PalmOS \end{array} \\ \begin{array}{c} WinMedia \\ Kinoma \end{array} & \begin{pmatrix} 1 & \infty \\ \infty & 1 \end{pmatrix} \end{array}$$

the above matrix shows the impacts of two operating systems (the top line) on two multimedia players (the left most column). The values in the matrix mean the Windows Media works fine in the WinCE operating system (WinCE) [59] but not in PalmOS, while Kinoma player [29] runs well in PalmOS instead of WinCE. The value of ratios does not have to be an integer. Suppose now we are about to find the better one in terms of the computing time from these two players on WinCE platform. We get the time value using the linear method as, for instance, 5 s for WinMedia and 2 s for Kinoma. Without the normalized matrix, Kinoma will be chosen as the better player; however, the fact is that Kinoma cannot run on WinCE at all. To get the correct result, we can use the first column of this normalized matrix to adjust the linear results by multiplying 2 s with ratio 1 for WinMedia and multiplying 5 s with ratio ∞ for Kinoma. Then the computing time of Kinoma becomes ∞, which immediately disqualifies itself.

$$PAD_{traffic} = \begin{pmatrix} pad_1^{traffic} \\ pad_2^{traffic} \\ \vdots \\ pad_n^{traffic} \end{pmatrix}, \quad PAD_{size} = \begin{pmatrix} pad_1^{size} \\ pad_2^{size} \\ \vdots \\ pad_n^{size} \end{pmatrix},$$

$$PAD^{client}_{comp} = \begin{pmatrix} pad_1^{cli-comp} \\ pad_2^{cli-comp} \\ \vdots \\ pad_n^{cli-comp} \end{pmatrix}, \quad PAD_{total} = \begin{pmatrix} pad_1^{total} \\ pad_2^{total} \\ \vdots \\ pad_n^{total} \end{pmatrix},$$

$$PAD^{server}_{comp} = \begin{pmatrix} pad_1^{svr-comp} \\ pad_2^{svr-comp} \\ \vdots \\ pad_n^{svr-comp} \end{pmatrix}, \qquad (1)$$

$$\mathcal{A} = \begin{array}{cc} & \begin{array}{ccc} cpu_0 & \dots & cpu_a \end{array} \\ \begin{array}{c} pad_0 \\ \vdots \\ pad_n \end{array} & \begin{pmatrix} \alpha_{00} & \dots & \alpha_{0a} \\ \vdots & \ddots & \vdots \\ \alpha_{n0} & \dots & \alpha_{na} \end{pmatrix} \end{array},$$

$$\mathcal{B} = \begin{array}{cc} & \begin{array}{ccc} os_0 & \dots & os_b \end{array} \\ \begin{array}{c} pad_0 \\ \vdots \\ pad_n \end{array} & \begin{pmatrix} \beta_{00} & \dots & \beta_{0b} \\ \vdots & \ddots & \vdots \\ \beta_{n0} & \dots & \beta_{nb} \end{pmatrix} \end{array},$$

$$\mathcal{R} = \begin{array}{c} \\ pad_0 \\ \vdots \\ pad_n \end{array} \begin{array}{c} ntwk_0 \;\dots\; ntwk_r \\ \begin{pmatrix} \gamma_{00} & \cdots & \gamma_{0r} \\ \vdots & \ddots & \vdots \\ \gamma_{n0} & \cdots & \gamma_{nr} \end{pmatrix} \end{array}, \tag{2}$$

$$\begin{aligned} PAD_{total} = {} & \frac{PAD_{size}}{Cli_{bandwidth} * \rho} + PAD_{comp}^{server} \\ & + \frac{Std_{cpu}}{Cli_{cpu}} \begin{pmatrix} \alpha_{0i} & 0 & \dots & 0 \\ 0 & \alpha_{1i} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \alpha_{ni} \end{pmatrix} \\ & \times \begin{pmatrix} \beta_{0j} & 0 & \dots & 0 \\ 0 & \beta_{1j} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \beta_{nj} \end{pmatrix} PAD_{comp}^{client} \\ & + \frac{Std_{bandwidth}}{Cli_{bandwidth}} \begin{pmatrix} \gamma_{0k} & 0 & \dots & 0 \\ 0 & \gamma_{1k} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \gamma_{nk} \end{pmatrix}. \\ & \times PAD_{traffic}. \end{aligned} \tag{3}$$

We abstract Eq. (3) to describe the total overhead for each PAD in one client environment, PAD_{total}, in Eq. (1). Cli_{cpu} is the device CPU speed in MHz, Cli_{mem} is the device memory size in MB, and $Cli_{bandwidth}$ is the client network bandwidth in Kbps. They come from the *DevMeta* from the client. On the right hand side of Eq. (3), the first part is the overhead of downloading the PAD. The parameter ρ is used to capture the available application level bandwidth in a real network deployment. It is usually between 0.6 and 0.8, depending on different network types. Based on our observation, we approximate ρ as 0.8 in our design. The second part is the computing overhead on the server side. This matrix can be achieved by pretesting each PAD on the application server. The third part is the computing overhead of running PAD on the client side. Suppose one client uses processor type i, operating system type j, and network type k, the algorithm finds the corresponding ratio vector $\left(\alpha_{0i}\ \alpha_{1i}\ \dots\ \alpha_{ni}\right)^{\mathrm{T}}$, $\left(\beta_{0j}\ \beta_{1j}\ \dots\ \beta_{nj}\right)^{\mathrm{T}}$, and $\left(\gamma_{0k}\ \gamma_{1k}\ \dots\ \gamma_{nk}\right)^{\mathrm{T}}$ from $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{R}$ based on its processor, operating system and network types. Given that we have only a limited number of consumer-used processors, OSes, and network types, the vector will be found with high probability. Otherwise a similar type with close parameters will be chosen instead. Then these vectors are extended to diagonal matrices, which are plugged into the Eq. (3) to adjust the linear estimation. The last part of the equation is the transmission overhead of running the PAD.

After we define the approach to calculate the total overhead of the PAD, the adaptation path search algorithm starts the first step by marking each node in the PAT with the total overhead computed by Eq. (3). An example is shown in Fig. 5. The number beside each node is the estimated total overhead. Infinity means that the PAD is not suitable for this client environment. Then the algorithm uses the Depth-First-Search-like algorithm to traverse each path from root to leaves and finds the path with the least sum of each PAD's total overhead. The PADs on this path are the negotiated protocol result for this client. The pseudo-code of the algorithm is shown in Fig. 6. Take Fig. 5 as an example, after line 3 in the pseudo-code in Fig. 6, the algorithm finishes marking each node with the total overhead shown as the number beside each node, the first path it examines is PAD1 and PAD4 and gets the *Least_Totoal_overhead* as 14 in code line 17, which is the selected shortest path so far, but when the algorithm searches along PAD2 and PAD7 with the *Least_Totoal_overhead* as 9, this new path becomes the shortest path and remains until the end of the search. Finally PAD2 and PAD7 form the final output path of the algorithm.

```
1   APSA (  IN: DevMeta, NtwkMeta, PADMeta ... PADMeta
2           OUT: PADMeta stack )
3   {       Use DFS to mark each node with its total overhead;
4           Least_Total_overhead = ∞;
5           path_total_overhead = 0;
6           v = root;
7           Create stack s;
8           s.push (v);
9           path_total_overhead += v.overhead;
10          mark v as visited;
11          while ( ! s.isEmpty() )
12          {
13                  if (no unvisited nodes are adjacent to the node on
14                     the top of the stack )
15                  {       if ( node on top of the stack is leaf
16                             && Total_overhead < Least_Total_overhead )
17                          {       Least_Total_overhead = Total_overhead;
18                                  PADMeta stack = s;
19                          }
20                          s.pop ();
21                          path_total_overhead -= node on the top.overhead;
22                  }
23                  else
24                  {       select an unvisited node u adjacent to the node
25                          on the top of the stack;
26                          s.push (u);
27                          path_total_overhead += u.overhead;
28                          mark u as visited;
29                  } // end if
30          } // end while
31  }
```

Fig. 6. The pseudo-code of the adaptation path search algorithm.

3.5. Mobile code security

PAD, the protocol adaptor, is the key element of the Fractal framework, and is implemented using mobile code. Security is a serious concern when deploying and running the PADs across heterogeneous environments, because the executable mobile code could possibly be written by a malicious user and allow an attacker to run native code that is subject to neither restrictions nor access control on the executing machine. In Fractal there are two techniques for securing PADs. First, sandbox [19], also known as virtual machine monitor techniques (VMM) [51], is needed to limit the privileges of PADs. The second technique used in Fractal is to assure that the source of the PAD is trustworthy using code-signing [38], in which the client manages a list of entities that it trusts. When a PAD is received, the client verifies that it was signed by an entity on this list. More advanced security techniques can be applied here, but it beyond the scope of this paper.

4. System capacity performance analysis

Now we are in a position to study the general system capacity of the proposed framework. Two case studies will be presented in the following two sections. In the Fractal framework, the adaptation negotiation and PAD downloading are two mandatory procedures for any protocol adaptation. Performance of the Fractal framework is greatly determined by the negotiation delay and download delay. In Eq. (3), the negotiation delay, which is the time between `INIT_REQ` and `PAD_META_REP` in Fig. 4 for each client, is not included because the negotiation time is not only related to the PAD itself but also to the protocol adaptation topology as well as the workload of the adaptation proxy. The PAD download time is also related to the CDN edge servers and the PAD size as well.

In this section, we first examine the negotiation time of four PADs of a one-level PAT as shown in Fig. 12 in Section 6. Here, all we care about is the downloading time of these four different PADs. The meaning of each node (i.e., PAD) of the PAT will be explained in the second case study in Section 6 later. We utilize some nodes from PlanetLab [44], specifically, the nodes at Wayne State University, New York University, and University of California at Berkeley, to emulate the CDN edgeservers as the distributed PAD servers to evaluate the PAD download time. PlanetLab has been accepted as a good platform to deploy academic-oriented CDNs platforms, such as CoDeeN [57] and Coral [16]. The adaptation proxy is set up at Wayne State University. Up to 300 clients access this adaptation proxy from the same network domain. Fig. 7(a) shows the average negotiation time of up to 300 clients sharing one adaptation proxy, denoted as AP on the y-axis. The x-axis is the number of clients. The y-axis represents the average negotiation time. Although some fluctuations occur, most of the negotiation times are between 20 to 27 ms. We also show the mean and median line of the measurement data in the figure. Given the fact that PlanetLab is a real overlay network built on top of Internet, it is quite normal to see this magnitude of fluctuation. The overall negotiation time remains in a relatively stable range for two reasons. First is the efficiency of the adaptation path search algorithm and the topology simplicity of the examined PAT. Second is that each client only needs one time negotiation in the same environment and the application session.

In order to show the benefit of deploying PADs on CDN edgeservers, we compare the average PAD retrieval time in two scenarios: *the centralized case*, in which up to 300 clients connect to a centralized PAD server in the same domain as the clients to download the PAD concurrently, and *the distributed case*, where the request traffic from the same number of clients is balanced to the three distributed PAD servers on PlanetLab to emulate the CDN edgeservers. The centralized PAD server is located at Wayne State University. The distributed PAD servers are located at New York University and University of California at Berkeley. On each server, there is a full set of PADs. Fig. 7(b) shows the average retrieval time to the number of clients in two scenarios. We can see clearly that the average PAD retrieval time rapidly goes up with the increasing number of clients in the centralized PAD server scenario. While for the distributed PAD server scenario, the retrieve time climbs up very slow with the increase of the number of clients. The bigger the number of client is, the more advantage can be taken from the distributed PAD servers in terms of the PAD retrieve time. To this end, we argue that using existing CDN infrastructure for PAD delivery is a very scalable and promising approach.

Table 1
The key length and size of each PAD used in the experiment (Case Study 1)

PAD name	Key length (bits)	Size (KB)
3DES	64	24
3DES	128	24
3DES	192	24
AES	128	21
AES	192	21
AES	256	21
RC4	64	10

5. Case Study 1: adaptive message encryption protocol

After studying the system capacity performance, we implement an adaptive message encryption protocol to evaluate the effectiveness and efficiency of the proposed Fractal framework. In this case study, we design an adaptive message encryption protocol between two communication parties: *a message sender* and *a message receiver*. We assume that some receivers use legacy applications, which support only old encryption algorithms, while some receivers have more flexibility to choose different algorithms. Three encryption algorithms, DES [9], AES [1], RC4 [46] are the candidates of encryption algorithms. We will show how the sender side adopts the Fractal framework to choose proper encryption algorithms based on their diverse characteristics and different client applications configurations. Note that this paper focuses on how to choose different algorithms in the context of symmetric encryption. The procedure to set up the symmetric key(s) is beyond the scope of this paper. It is very easy to set up the symmetric keys using the Diffie and Hellman [11] key exchange or certificate-based authentication. First we will give a brief introduction about the three encryption algorithms used in this case study.

5.1. Three encryption algorithms

Many symmetric key encryption algorithms have been proposed. DES, AES, and RC4 are three of the most popular shared-key encryption algorithms.

(1) *DES/Triple DES* [9] Data Encryption Standard is addressed in FIPS PUB 46. Data are encrypted in 64-bit blocks using a 56-bit key. DES transforms 64-bit input in a series of steps into a 64-bit output. The same steps and the same

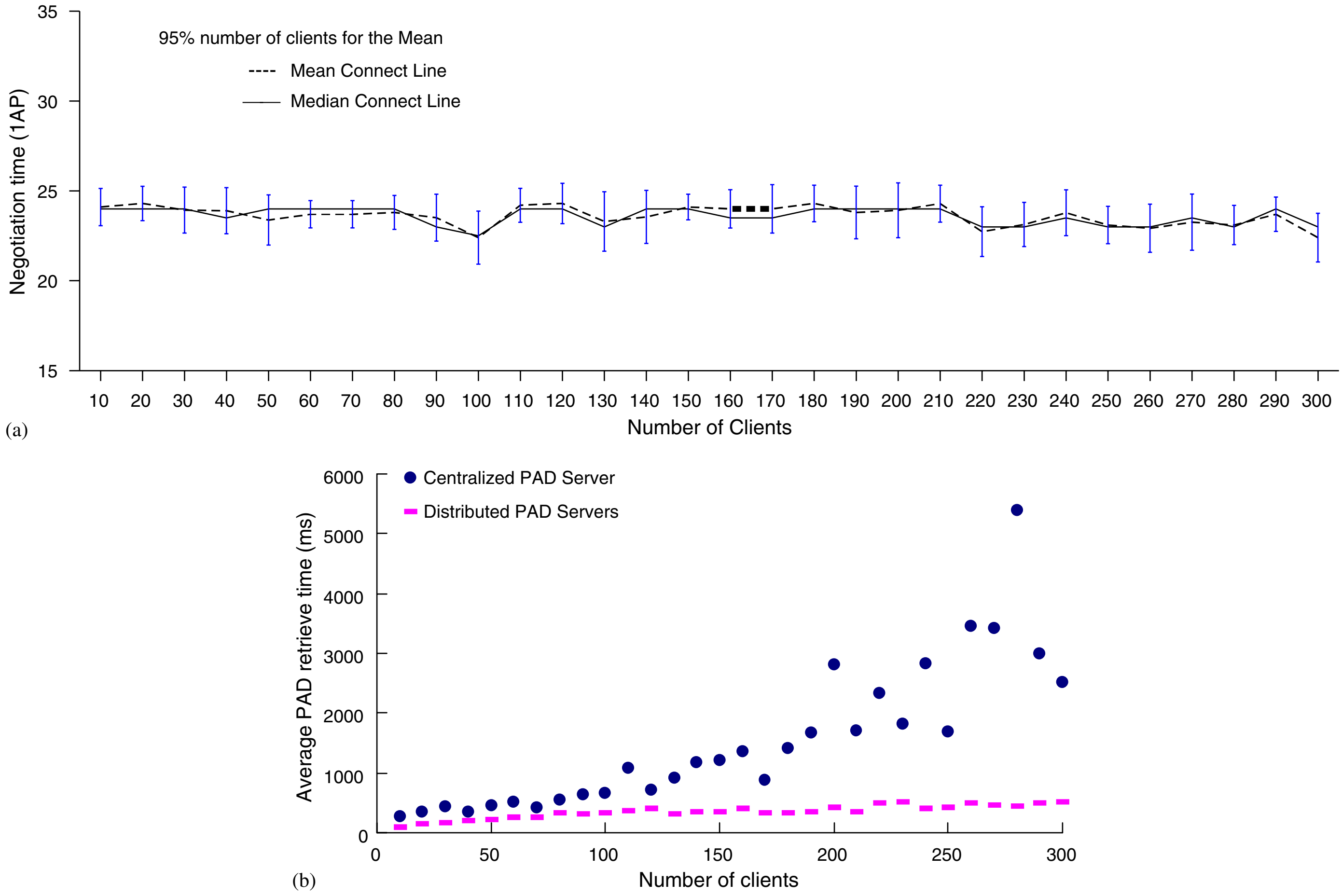

Fig. 7. System capacity analysis: (a) the average negotiation time; (b) the average PAD retrieval time from centralized and distributed PAD servers.

key are used to decrypt the data. With the development of hardware technology, DES shows potential vulnerability to a brute-force attack. Triple DES (3DES) is an alternative of traditional DES algorithm. National Institutes of Standards and Technology (NIST) requires all new applications should use triple DES or more advanced encryption algorithms, while DES is still supported for legacy applications.

(2) *AES* [1] AES is a relatively new algorithm compared with DES. Observing that DES is more and more out of date and 3DES is not a long term replacement candidate for widely used the DES algorithm. NIST called a new Advanced Encryption Standard (AES). AES is more secure than DES. It can has key length as long as 256 bits. It also have high computation efficiency and flexibility to be practical in a wide range of applications.

(3) *RC4 Stream Cipher* [46] RC4 is a contemporary variable key-size stream cipher with byte-oriented operations. It is based on the use of a random permutation. Key length is in a range from 1 to 256 bytes. RC4 is easy to be implemented even on resource-constraint devices, such as Berkeley Motes and smart cards. Adjustment of key length can achieve a tradeoff between running speed and security level.

There are several other symmetric algorithms have been proposed; however, we believe these three algorithms are diverse enough to show the basic idea of adaptive message encryption in this case study.

5.2. *Experimental platform*

In our experimental platform, as shown in Fig. 8, three kinds of client hosts, *desktop*, *laptop*, and *Pocket PC*, use different message receiver applications, to connect to the message sender and an adaptation proxy. The hardware and software configurations of the servers and clients are also shown in Fig. 8. The message sender has 100 messages with size as 100 KB. We implement three encryption algorithms, 3DES, AES and RC4 in C code as three PADs. The first two encryption algorithms has three different key length settings. Key length and size of each algorithm is shown in Table 1. We also implement an adaptation proxy connected with the application server in the same LAN domain. To emulate the behavior of the real content distribution network and edgeservers, we utilize three nodes from PlanetLab [44] as the distributed PAD servers, similar to the previous section.

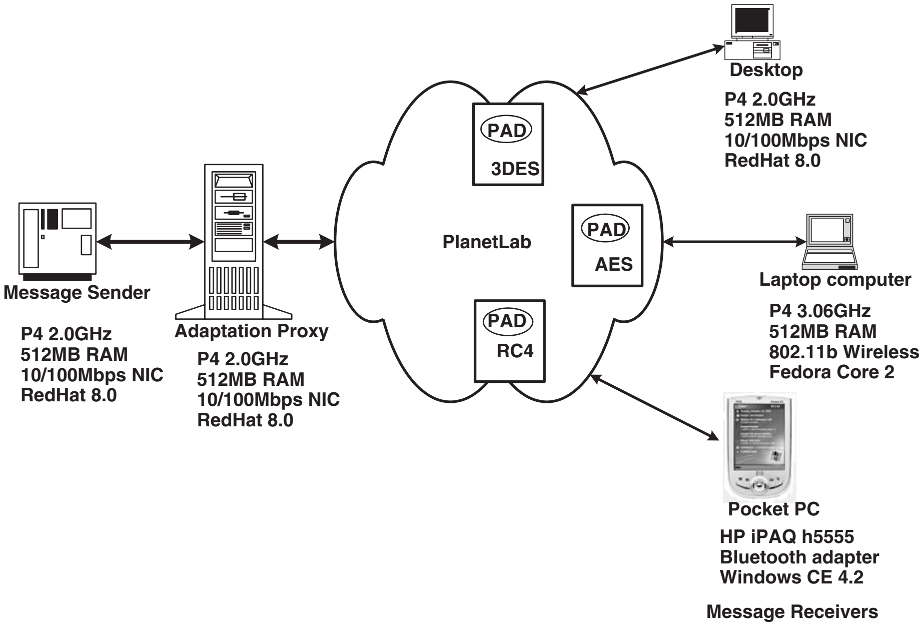

Fig. 8. Configurations of the experimental platform (Case Study 1).

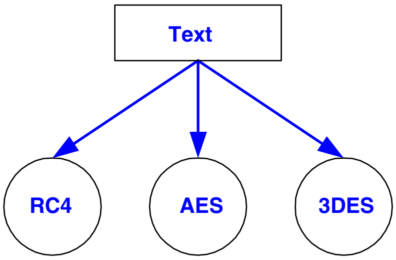

Fig. 9. Protocol adaptation tree of the adaptive message encryption (Case Study 1).

Table 2
The average download time in milliseconds of each PAD from different PlanetLab nodes

PAD name	Wayne State University	New York University	UC Berkeley
3DES	2.24	3.22	3.64
AES	2.25	3.12	3.58
RC4	2.23	3.20	3.84

5.3. Experimental adaptation model

Following the Fractal framework, we first define the PADs used in this application and construct the PAT for this case study, as shown in Fig. 9. The PAT in this case study is a one-level tree. Each leaf is an encryption PAD that can be used on a specific message receiver environment to reduce the total delay overhead between the sender application and receiver application. Then we follow Eq. (3) to generate the specific Eq. (10) for this case study. We use $pad^{total}_{3DES-64}$, $pad^{svr-comp}_{3DES-64}$ and $pad^{cli-comp}_{3DES-64}$ to represent three parameters of *3DES* PAD with 64 bits key: the total time overhead is defined as the time from the start of downloading the PAD to the end of the application session, the server side computing overhead, and the client side computing overhead. For other PADs the definitions are similar. Table 2 shows the average download time in milliseconds of each PAD from three PlanetLab nodes in three research institutions located at middle, east, and west of North America. The size of those PADs are from 10 to 25 KB. As we can see from the Table, the downloading time from New York University and UC Berkeley are roughly the same. It is slightly faster from Wayne State University since we do our experiment in the same network domain. Given that the instability of the PlanetLab, we consider the PAD download time from all distributed nodes are similar, following the same pattern we found in the Section 4. On the other hand since the input of these encryption algorithms has the same size as the output, the traffic overhead (i.e., the bandwidth requirements) incurred by each algorithm for different networks are also identical. So in the evaluation of the total delay time in Eq. (3) we exclude the PAD download time and traffic overhead because they are pretty much the same. In the next case study, we will examine the impact of the PAD downloading time and traffic overhead which we do not analyze in this case study.

For the adaptive message encryption case study we first introduce the normalized ratio matrix $\mathcal{A}$ as listed in Eq. (4), based on the definition of Eq. (2) in the Fractal framework. In Eq. (4), *P*, *D*, and *L* represent the Intel PXA 255 processor in Pocket PC, Pentium IV 2.0 GHz processor in Desktop, and Pentium IV 3.06 GHz processor in Laptop, respectively. Because most of the operations in these encryption algorithms are bit operations instead of float-point operations, they have almost same running efficiency in these client CPU types. We set all values as 1. Network normalized ratio matrix is not included since we will not evaluate the network overhead in the case study. Instead, we use normalized ratio matrix ($\mathcal{B}$) as listed in Eq. (5) to demonstrate the different encryption requirements of message receiver applications. For example, the legacy systems only use the DES algorithm while the new applications will utilize the new encryption algorithms. Correspondingly in the normalized ratio matrix, we set the ratio as 1 for 3DES algorithm and ∞ for others in legacy systems. In our experimental platform, we specify the receiver applications on desktop as a legacy system and that on laptop and PocketPC as a new system. This may not be always true in reality, but it is enough to show the point of adaptation in this case study.

For a newcoming request from a message receiver application (i.e., a client), Fractal will find the receiver's processor type and application type from the device metadata, such as, i (for processor type) and j (for application type). Then the normalized ratio matrix can be formed by collecting corresponding columns at $\mathcal{A}(i)$ and $\mathcal{B}(j)$. Finally with other available receiver side metadata, the total time overhead of each PAD for this new receiver can be computed using Eq. (6). After obtaining the total time overhead of each PAD, we will run the adaptive path search algorithm against them and choose the most proper encryption algorithm for a specific client. Next, we describe the performance evaluation results.

$$\mathcal{A} = \begin{array}{cc} & \begin{array}{ccc} P & D & L \end{array} \\ \begin{array}{c} \mathit{3DES-64} \\ \mathit{3DES-128} \\ \mathit{3DES-192} \\ \mathit{AES-128} \\ \mathit{AES-192} \\ \mathit{AES-256} \\ \mathit{RC4-64} \end{array} & \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}, \tag{4}$$

$$\mathcal{B} = \begin{array}{cc} & \begin{array}{cc} \mathit{LegacySystem} & \mathit{NewSystem} \end{array} \\ \begin{array}{c} \mathit{3DES-64} \\ \mathit{3DES-128} \\ \mathit{3DES-192} \\ \mathit{AES-128} \\ \mathit{AES-192} \\ \mathit{AES-256} \\ \mathit{RC4-64} \end{array} & \begin{pmatrix} 1 & \infty \\ 1 & \infty \\ 1 & \infty \\ \infty & 1 \\ \infty & 1 \\ \infty & 1 \\ \infty & 1 \end{pmatrix} \end{array}, \tag{5}$$

$$\begin{pmatrix} pad^{total}_{3DES-64} \\ pad^{total}_{3DES-128} \\ pad^{total}_{3DES-192} \\ pad^{total}_{AES-128} \\ pad^{total}_{AES-192} \\ pad^{total}_{AES-256} \\ pad^{total}_{RC4-64} \end{pmatrix} = \begin{pmatrix} pad^{svr-comp}_{3DES-64} \\ pad^{svr-comp}_{3DES-128} \\ pad^{svr-comp}_{3DES-192} \\ pad^{svr-comp}_{AES-128} \\ pad^{svr-comp}_{AES-192} \\ pad^{svr-comp}_{AES-256} \\ pad^{svr-comp}_{RC4-64} \end{pmatrix}$$
$$+\frac{cpu}{Cli_{cpu}}$$
$$\times \begin{pmatrix} \alpha_{3DES-64(i)} & & & & & & 0 \\ & \alpha_{3DES-128(i)} & & & & & \\ & & \alpha_{3DES-192(i)} & & & & \\ & & & \alpha_{AES-128(i)} & & & \\ & & & & \alpha_{AES-192(i)} & & \\ & & & & & \alpha_{AES-256(i)} & \\ 0 & & & & & & \alpha_{RC4-64(i)} \end{pmatrix}$$
$$\times \begin{pmatrix} \beta_{3DES-64(j)} & & & & & & 0 \\ & \beta_{3DES-128(j)} & & & & & \\ & & \beta_{3DES-192(j)} & & & & \\ & & & \beta_{AES-128(j)} & & & \\ & & & & \beta_{AES-192(j)} & & \\ & & & & & \beta_{AES-256(j)} & \\ 0 & & & & & & \beta_{RC4-64(j)} \end{pmatrix}$$
$$\times \begin{pmatrix} pad^{cli-comp}_{3DES-64} \\ pad^{cli-comp}_{3DES-128} \\ pad^{cli-comp}_{3DES-192} \\ pad^{cli-comp}_{AES-128} \\ pad^{cli-comp}_{AES-192} \\ pad^{cli-comp}_{AES-256} \\ pad^{cli-comp}_{RC4-64} \end{pmatrix}. \tag{6}$$

5.4. *Performance evaluation of encryption algorithms adaptation*

We test the total time overhead of each algorithm for message receivers on desktop, laptop, and PocketPC, as shown in Fig. 10. The x-axis lists different encryption algorithms, the y-axis shows the total time for each algorithm including the sender encryption time and the receiver decryption time. In Fig. 10(a), since the receiver application of the desktop is a legacy application in our experimental setup, which accepts only DES algorithms, the output of the adaptive path selection algorithm of Fractal will set all other encryption algorithms except DES algorithms to infinite, which is denoted as N/A in the figure. However, for comparison purpose, we also show their corresponding computing overhead on the same figure. As a matter of fact, although AES-class algorithms have less computing overhead, they will not be chosen as the proper encryption algorithm for the desktop, which runs legacy applications only. Now only 3DES algorithms are eligible candidates. It is trivial that 3DES with 64 bits key should run faster than 3DES with 128 bits or 192 bits length key. Usually the Fractal framework will recommend the receiver to choose 3DES-64 since it has the fastest running speed with reasonable security enforcement. But this does not prevent application from choosing 128 or 192 bits 3DES. By introducing more adaptation parameters, like a normalized matrix for application security requirements, more secure algorithm could be selected. We believe this is a trivial task and decide not to dig inside in this paper.

For the applications running on the laptop, 3DES is obviously not considered because it is out of date (and replaced by AES algorithms) for new applications. AES-128 which has slightly

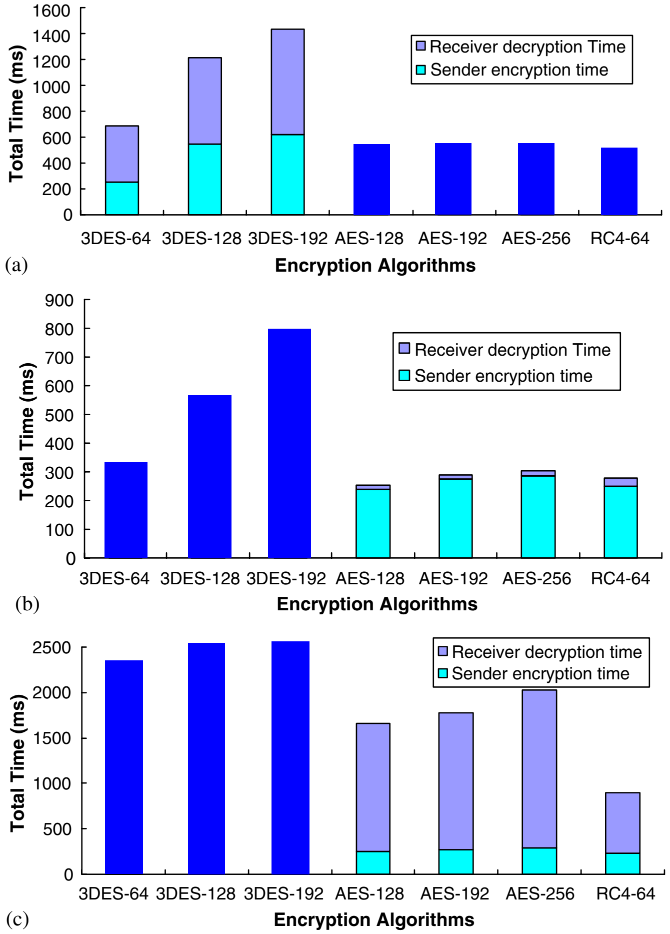

Fig. 10. A comparison of the total time overhead for different message receivers: (a) Desktop; (b) Laptop; and (c) PocketPC.

less total time overhead than other three algorithms have, as shown in Fig. 10(b), will be selected by the Fractal framework. Note that similar to the case for desktop, other AES algorithms could also be selected for more secure purpose by extending the total time overhead evaluation formula. Finally, in Fig. 10(c), we can see that the major part of the total time overhead is contributed by the receiver decryption time because the hardware of PocketPC on which receiver application executes is not as powerful as desktop or laptop hardware configurations. Not surprising, Fractal selects RC4-64 as the most appropriate encryption algorithm, which is much faster than other algorithms. This is compatible with the fact that RC4 is almost the default encryption algorithm for small resource-constraint devices. It is worth noting that the choice made by the Fractal framework is straightforward in this case study. However, our work is the first effort to make the choice making in a formal way. We believe that the Fractal framework will be more useful in complicated applications in the foreseeable future, as shown in the second case study. Our future work includes investigating more encryption algorithms in heterogeneous environments, and applying this technique to the distributed computer-assistant surgery application [33].

6. Case Study 2: adaptive communication optimization protocol

We have seen in the previous two sections the system capacity performance and the adaptation effectiveness in the message encryption adaptation case study. However, two metrics are omitted in the first case study: the PAD download time and traffic overhead. We add these two metrics into the second case study in which we implement an adaptive communication optimization protocol prototype. The Fractal framework also shows good performance in this more complicated scenario. The basic idea of the adaptive communication optimization is to dynamically select different communication protocols, including `Direct sending`, `Gzip`, `Vary-sized blocking` [37], `Bitmap` [33], to adapt to different network conditions. This application is motivated by our recent analysis of four different communication optimization algorithms [32], in which we found that different communication optimization techniques exhibit different performance in different network environments as well as for different document types. These techniques are good examples of protocols that reduce the overall communication overhead, and inspire us to use this case to test Fractal. In the following context, we first briefly introduce each communication optimization protocol, followed by experiment platforms, the specific protocol adaptation model, and result analysis.

6.1. Four communication optimization protocols

Several application-specific optimization techniques have been proposed in different contexts. Generally, they work in two fashions to reduce bandwidth requirement. One is to compress content at the server side and decompress at the client side. The other is to calculate the difference between old and new versions of the content on the server side, send difference to the client and rebuild the new version based on the difference received by the client and the old version that the client already had. In the section, we examine the four communication optimization protocols used in our case study.

(1) *Direct sending*: In this protocol, strictly speaking, there is no communication optimization technique, client and Web server just directly send content to each other. In this simple case, the client still needs to negotiate with the adaptation proxy at the beginning.
(2) *Gzip*: In this algorithm, we use gzip to compress the Web page at the Web server and decompress it at the client side. Gzip is a popular data compression program [23] which uses the LZ77 algorithm.
(3) *Vary-sized blocking*: Proposed in LBFS [37] for reducing traffic further, the idea of LBFS is that of content-based chunk identification. Files are divided into chunks, demarcated by points where the Rabin fingerprint [45] of the previous 48 bytes matches a specific polynomial value. This tends to identify portions even after insertions and deletions have changed its position in the file. The boundary regions

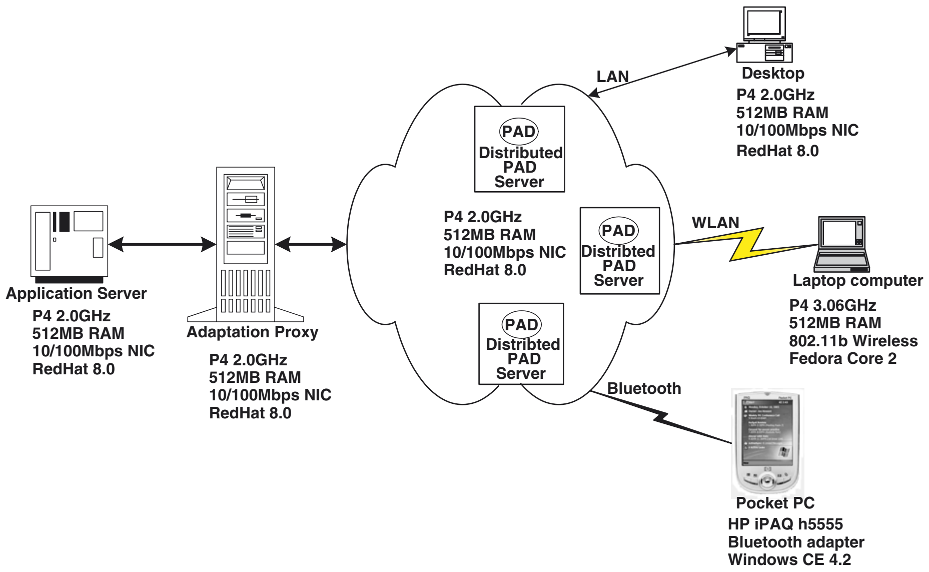

Fig. 11. Configurations of the experimental platform (Case Study 2).

Table 3
The functions and implementations of PADs used in the experiments (Case Study 2)

PAD name	Function	Implementation
Direct	Null	Null
Gzip	Compression	Java class object
Vary-sized blocking	Differencing files using Fingerprint	Java class object
Bitmap	Differencing files bit by bit	Java class object

are called breakpoints. The server generates the difference between two versions of a file by comparing the digest of each chunks and saves the different chunks. It is powerful to reduce the size of the difference but with expensive computing overhead on both sides. *Vary-sized blocking* has been adopted by several projects as well [8,12,35,53].

(4) *Bitmap*: Proposed in [33], the idea behind *Bitmap* is that files are updated by dividing both files into fix-sized chunks. The client sends digests of each chunk to the server, and the server responds only with new data chunks. Based on the old version and the differencing, the new version can be rebuilt. It has outperforming results compared with other differencing algorithms for some image formats like DICOM [10], BMP, and so forth.

Basically, our evaluation results in [32] show that no single algorithm outperforms others in all cases. Different approaches have different performance in terms of different metrics. A completely different result can be achieved by the same algorithm when it is applied against different types of documents. Network bandwidth affects the performance of algorithms substantially as well. The performance can also be influenced by different parameter settings of the same algorithm. More details can be found in [32].

6.2. Experimental platform

Fig. 11 shows the experimental platform, where three kinds of client hosts, *desktop*, *laptop* and *PocketPC*, use three types of network connections, *LAN*, *Wireless LAN* and *Bluetooth*, to connect to an application server and an adaptation proxy. Same as the first case study, the hardware and software configurations of the servers and clients are also shown in Fig. 11. The application server holds a set of 75 Web pages with the average size of about 135KB consisting of 5KB text and four images totalling about 130KB, which is inspired by a typical example of a medical application server that holds four images of different 3D views [33]. We use Java to implement four communication optimization techniques as four PADs. The summary of function and implementation of each PAD is shown in Table 3. We also implement an adaptation proxy connected with the application server in the same LAN domain. To emulate the behavior of the real content distribution network and edge-servers, similar as first case study, we still utilize some nodes from PlanetLab [44] as the distributed PAD servers. We set up a centralized PAD server which holds all the PADs for the purpose of performance comparisons between centralized and distributed PAD servers.

6.3. Experimental adaptation model

Guided by the Fractal framework as before, we first define the PADs used in this application and construct the PAT for this case study, as shown in Fig. 12. The PAT in this case study is also one-level tree. Each leaf is a communication optimization PAD that can be used on a specific client environment to reduce the total communication time between the client and the application server. Then we base on Eq. (3) to generate the specific Eq. (10) for this case study. We use pad_{direct}^{total}, pad_{direct}^{size}, $pad_{direct}^{svr-comp}$, $pad_{direct}^{cli-comp}$, and $pad_{direct}^{traffic}$ to represent five parameters

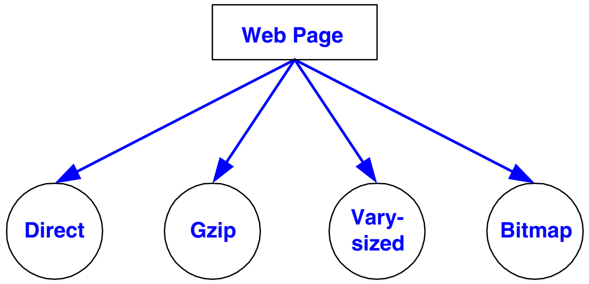

Fig. 12. Protocol adaptation tree of the adaptive communication optimization (Case Study 2).

of *Direct sending* PAD: the total time overhead defined as the time from the start of downloading the PAD to the end of the application session, the size of the PAD, the server side computing overhead, the client side computing overhead, and the traffic overhead generated by the PAD. For other PADs the definitions are similar. Note that protocols like *Vary-sized blocking* and *Bitmap* have to compute the difference on the server side and rebuild a new version on the client side to reduce the bandwidth requirement. In this case study we use all of the normalized ratio matrix proposed in Fractal framework, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{R}$ in Eqs. (7), (8), and (9). In Eq. (7), P, D, and L have the same meaning as they have in the first case study. Some of the data (e.g., 1.1) come from the test results, others we set as 1 to follow the linear model.

For a newcoming client, Fractal will find its processor type, OS type, and network type, such as, i, j, and k. Then the normalized ratio matrix can be formed by collecting corresponding columns at $\mathcal{A}(i)$, $\mathcal{B}(j)$, and $\mathcal{R}(k)$. Finally with other available client side metadata, the total time overhead of each PAD for this new client can be computed using Eq. (10).

$$\mathcal{A} = \begin{array}{c} \\ direct \\ gzip \\ vary \\ bitmap \end{array} \begin{array}{c} P\ \ D\ \ L \\ \begin{pmatrix} 1 & 1 & 1 \\ 1.1 & 1 & 1 \\ 1.1 & 1 & 1 \\ 1.1 & 1 & 1 \end{pmatrix} \end{array} \tag{7}$$

$$\mathcal{B} = \begin{array}{c} \\ direct \\ gzip \\ vary \\ bitmap \end{array} \begin{array}{c} WinCE4.2\ \ FedoraCore2 \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \end{array} \tag{8}$$

$$\mathcal{R} = \begin{array}{c} \\ direct \\ gzip \\ vary \\ bitmap \end{array} \begin{array}{c} LAN\ \ WLAN\ \ Bluetooth \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array} \tag{9}$$

$$\begin{aligned} \begin{pmatrix} pad_{direct}^{total} \\ pad_{gzip}^{total} \\ pad_{vary}^{total} \\ pad_{bitmap}^{total} \end{pmatrix} &= \frac{1}{Cli_{bandwidth}\alpha} \begin{pmatrix} pad_{direct}^{size} \\ pad_{gzip}^{size} \\ pad_{vary}^{size} \\ pad_{bitmap}^{size} \end{pmatrix} \\ &+ \begin{pmatrix} pad_{direct}^{svr-comp} \\ pad_{gzip}^{svr-comp} \\ pad_{vary}^{svr-comp} \\ pad_{bitmap}^{svr-comp} \end{pmatrix} \\ &+ \frac{cpu}{Cli_{cpu}} \\ &\times \begin{pmatrix} \alpha_{direct(i)} & & & 0 \\ & \alpha_{gzip(i)} & & \\ & & \alpha_{vary(i)} & \\ 0 & & & \alpha_{bitmap(i)} \end{pmatrix} \\ &\times \begin{pmatrix} \beta_{direct(j)} & & & 0 \\ & \beta_{gzip(j)} & & \\ & & \beta_{vary(j)} & \\ 0 & & & \beta_{bitmap(j)} \end{pmatrix} \\ &\times \begin{pmatrix} pad_{direct}^{cli-comp} \\ pad_{gzip}^{cli-comp} \\ pad_{vary}^{cli-comp} \\ pad_{bitmap}^{cli-comp} \end{pmatrix} \\ &+ \frac{bandwidth}{Cli_{bandwidth}} \\ &\times \begin{pmatrix} \gamma_{direct(k)} & & & 0 \\ & \gamma_{gzip(k)} & & \\ & & \gamma_{vary(k)} & \\ 0 & & & \gamma_{bitmap(k)} \end{pmatrix} \\ &\times \begin{pmatrix} pad_{direct}^{traffic} \\ pad_{gzip}^{traffic} \\ pad_{vary}^{traffic} \\ pad_{bitmap}^{traffic} \end{pmatrix}. \end{aligned} \tag{10}$$

6.4. Results of communication protocol adaptation

We test each client configuration in three adaptation scenarios: `No protocol adaptation`: There is no communication optimization protocol, the client connects to the Web server and

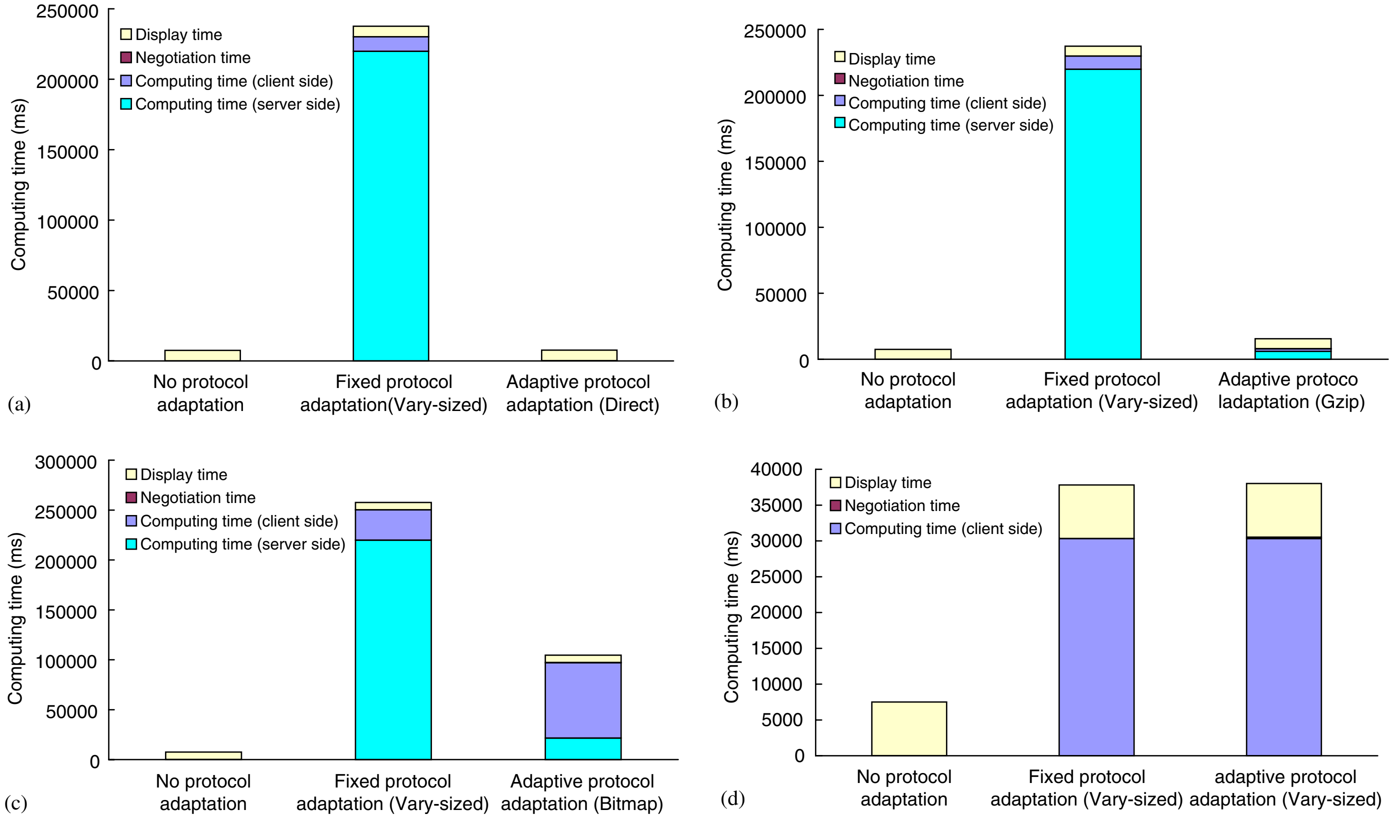

Fig. 13. A comparison of computing overhead in different environments: (a) Desktop in LAN with server side computing; (b) Laptop in Wireless LAN with server side computing; (c) PocketPC in Bluetooth with server side computing; and (d) PocketPC in Bluetooth without server side computing.

directly receives the original Web page; `Fixed protocol adaptation`: All clients always use one protocol, *Vary-sized blocking*, to talk with the Web server without the negotiation procedure with the adaptation proxy; `Adaptive protocol adaptation`: The full function of Fractal is utilized to do the protocol adaptation.

Fig. 13 shows the computing overhead in three adaptation scenarios for different client configurations. The horizontal line shows three adaptation scenarios with the selected protocol in the parentheses and the vertical line, representing the computing overhead, consists of several components respectively. Fig. 14(a) illustrates the bandwidth requirement in KBytes on the y-axis for each client environment as shown in the x-axis. We assume different clients perform identical application requests. The same protocol should generate the same number of bytes transferred, no matter the kind of client environment. First let us look at Fig. 13(a)–(c), which include both server side and client side computing overheads. The server side computing is used by the application server to dynamically encode the application content, e.g., compute the difference between two versions of Web pages. The client side computing overhead is used to decode the application content, e.g., rebuilding new version based on the difference and old version. *Vary-sized blocking* has huge server side computing time, which disqualifies it as the adaptive protocol for any of the client environment even if it generates the least transfer bytes as shown in Fig. 14(a). Different client configurations result in different negotiated protocols, such as *Direct sending* for desktop in LAN, *Gzip* for laptop in Wireless LAN, and *Bitmap* for PocketPC in Bluetooth.

We can see that *Gzip* in Fig. 13(b) and *Bitmap* in Fig. 13(c) have more or less unbalanced server and client side computing time. Since the overweighed server side computing time plays an important role in the total overhead for some protocols, e.g., *Vary-sized blocking*, different adaptation results may be observed if getting rid of the server side computing time from the total overhead. We pre-compute the server side computing tasks for each protocol on each Web page to exclude the server side computing overhead from the total computing time. We found that although the negotiated adaptation protocols for Desktop in LAN and Laptop in Wireless LAN remain the same, the adaptive protocol for PocketPC in Bluetooth changes from *Bitmap* to *Vary-sized blocking* as shown in Fig. 13(d). Note that the scale of (c) and (d) are one order of magnitude different. The difference in negotiation results again shows that our approach can adapt the protocol according to different application strategies as well as the client environments.

In Fig. 14(a), *Direct sending* generates the most traffic bytes while *Vary-sized blocking* has the least bytes transferred. *Gzip* and *Bitmap* are in the middle in terms of bytes transferred. Computing time and bytes transferred are two components of the total overhead. In fast networks the bytes can be transferred in small time slots so that the transmission time has a smaller effect on the total overhead than the computing time. But in slow networks, bytes transferred will result into a transmission time that outweighs the computing time and dominates the total overhead. So the comprehensive influence from these two factors forms the different total overhead time performance shown in Fig. 14(b) and (c). For each client configuration the adaptive protocol achieves the least total time, like *Gzip* for laptop in wireless, *Bitmap* for PocketPC in Fig. 14(b). In the same client configuration, adaptive protocol may vary according to different server strategies, for example *Vary-sized blocking* becomes the best choice for PocketPC in Bluetooth without server side computing as shown in Fig. 14(c). The adaptive protocols pointed by the oval in Fig. 14 are the best choices in different scenarios, which comply exactly with the negotiation results from Fractal.

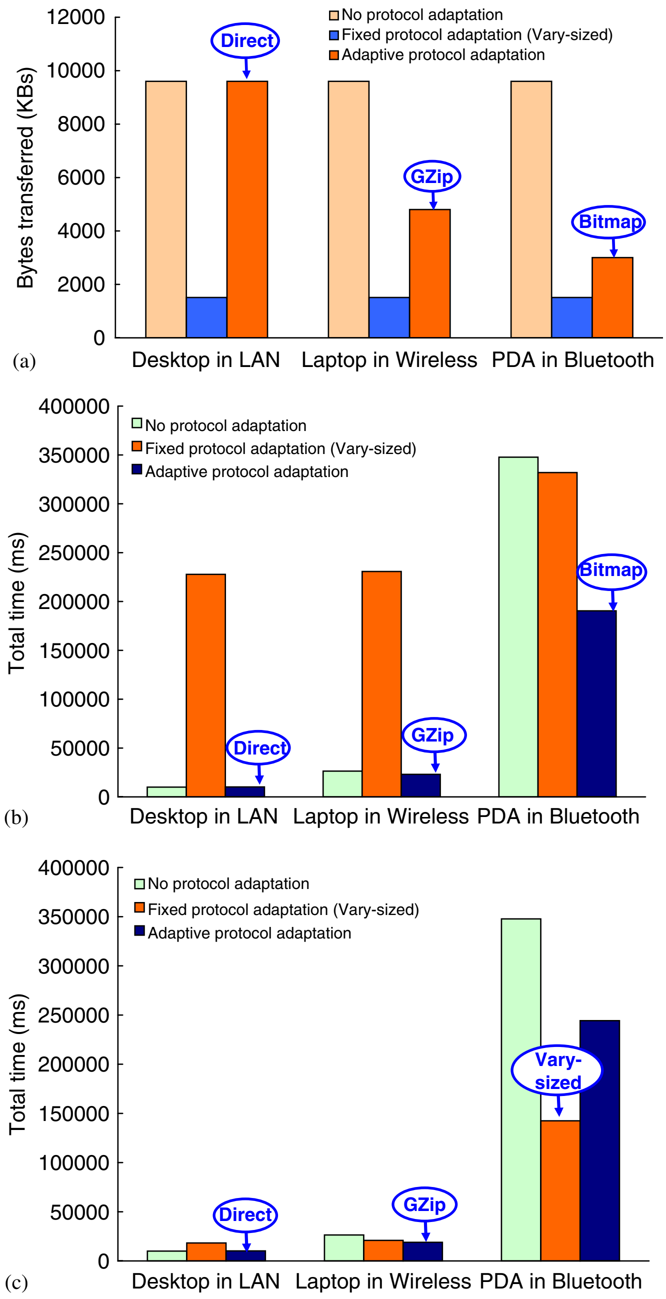

Fig. 14. Comparison of three different scenarios: (a) bytes transferred; (b) total time with server side difference computing; (c) total time without server side difference computing.

7. Related work and discussions

Fractal shares its goals with some recent efforts that are aimed at injecting functionality into application for adaptation. We categorize related research into four groups as *distributed adaptation*, *protocol adaptation*, *mobile code and mobile agent*, and *communication optimization*.

Distributed adaptation: From the Internet topology's point of view, adaptation functionality can be introduced either at the end-points or distributed on intermediate nodes. Odyssey [39], Rover [27] and InfoPyramid [36] are examples of systems that support end point adaptation. Conductor [60] and CANS [17] provide an application transparent adaptation framework that permits the introduction of arbitrary adaptors in the data path between applications and end services. While these approaches provide an extremely general adaptation mechanism, significant change to existing infrastructure is required for their deployment. However, Fractal solves the deployment problem by leveraging the existing CDNs technology to distributed PADs, which are implemented using mobile code.

From the network structure's perspective, there are two issues: whether adaptation functionality is introduced at network layer with application-transparency or at the application level with application-awareness. Systems such as transformer tunnels [49] and protocol boosters [34] are examples of application-transparent adaptation efforts that work at the network level. Such systems can cope with localized changes in network conditions but cannot adapt to behaviors that differ widely from the norm. Moreover, their transparency hinders composability of multiple adaptations. More general are programmable network infrastructures, such as COMET [6], which supports flow-based adaptation, and Active Networks [52,58], which permit special code to be executed for each packet at each visited network element. While these approaches provide an extremely general adaptation mechanism, significant change to existing infrastructure is required for their deployment. Fractal overcomes this shortcoming because it works entirely on the application level. Similar efforts also work at the application level. The cluster-based proxies in BARWAN/Daedalus [14], TACC [15], and MultiSpace [20] are examples of systems where application-transparent adaptation happens in intermediate nodes (typically a small number) in the network. Active Services [3] extends these systems to a distributed setting by permitting a client application to explicitly start one or more services on its behalf that can transform the data it receives from an end service. Fractal is different from other application level frameworks in the following ways: first, it is not using intermediate nodes which may occur with deployment problems. Second it does not rely on any specific data stream or client conditions. On the contrary, it is designed to cope with any applications and client environments as long as one has the proper protocol adaptor.

Protocol adaptation: There are some research work about the protocol adaptation. In network level systems such as [43], in which communicating end hosts use untrusted mobile code to remotely upgrade each other with the transport protocols that they use to communicate. Transformer tunnels [49] and protocol boosters [34] are doing application-transparent adaptation by tuning the network protocol according to the change of network situations. Such systems can deal with localized changes in network conditions but cannot react to changing environments outside the network layer. Since Fractal works at the application layer, it can maximally adapt application level protocols which have no way to be completed in the network layer. Fractal is also different from the Web browser plugins, e.g., Realplay, Flash, and so on. Plugin is an application component which completes part of the functionality, incapable of doing protocol adaptation. Although today some Web sites provide multiple choices of plugins to do the similar function, they still need the client to manually select one, but maybe not the best. Fractal is a general framework to adapting the functionality by means of protocol adaptation which has transparency to the client and other characteristics, such as flexibility and extendibility, which plugins do not have.

Mobile code and mobile agent: Mobile code is a good candidate for carrying a protocol module since it has long been known as a mechanism for providing a late binding of function to systems [5,26,28]. Mobile code and related technologies also have been proposed and studied as effective means of implementing content adaptation, protocol update, and program migration in distributed applications. In [42,43] they propose a system in which communicating end hosts use untrusted mobile code to remotely upgrade each other with the transport protocols that are used to communicate. Our work is complimentary to their work because our proposal works in the application level. A new lightweight, component-based mobile agent system that can adapt to diverse devices and features resource saving is proposed in [7]. In this system, mobile code is brought in and associated execution states of an application dynamically after migration. NWSLite [22] provides a sophisticated predicting tools for the remote code execution offloaded from mobile client to the close server. To our best knowledge, Fractal is the first framework to use mobile code to do protocol adaptation that extends the utilization of mobile code technology.

Communication optimization: As far as the communication optimization techniques go, *Fix-sized blocking* was used in the Rsync [54] software to synchronize different versions of the same data. In this approach, files are updated by dividing both files into fix-sized chunks. The client sends digests of each chunk to the server, and the server responds only with new data chunks. *Vary-sized blocking* was proposed in LBFS [37] for further reducing traffic. Recently, several projects such as CASPER wide-area file system [53], Pond prototype [47], and Pastiche backup system [8], adopt vary-sized blocking to either improve the system performance or reduce the storage requirements. Our work compliments these efforts, and the result of this paper can be applied in their work directly. Spring and Wetherall have proposed a protocol independent technique for eliminating redundant network traffic [48]. When one end wants to send data that already exists at other end, it instead sends a token specifying where to find the data at the other end.

A lot of encryption algorithms have been proposed, e.g., DES [9], AES [1], and RC4 [46], however, the focus of this paper is on selecting an appropriate encryption algorithm for a specific client configuration. Therefore, we envision our work complements to the research of cryptography algorithms very well. With the emergence of more and more application-level protocols, like encryption algorithms, communication optimization algorithms, adaptation becomes a necessity because each algorithm has distinct characteristics from others even all of them are for the same application purpose. Encryption algorithm family is a good example. Some of them are very secure but require more computing power. Some of them are very simple but can run on tiny devices. Communication optimization algorithms also justify this fact. We believe that our work makes an initial step towards using mobile code to support the application-level protocol adaptation, in which the protocol is composed of a series of PADs. These are packaged as mobile code modules and distributed by existing CDNs. Furthermore, Fractal provides a general framework for other adaptation functionality as well by extending the PAD into other adaptation functions, e.g., content adaptation.

8. Conclusions

In this paper, Fractal, a dynamic protocol adaptation framework, is proposed to benefit the application from choosing appropriate protocols according to dynamic client devices and network environments. To the best of our knowledge, this is the first effort on protocol adaptation by means of mobile code and CDNs edgeservers. An adaptive message encryption protocol and an adaptive communication optimization protocol have been built in the context of this framework. For adaptive message encryption protocol, the Fractal framework shows great flexibility in selecting a proper encryption algorithm. For the adaptive communication optimization protocol, performance comparison with other protocol adaptation approaches shows that Fractal has lightweight system overhead, small resource footprint, and noticeable client performance improvement. Our next step includes integrating Fractal with end to end service differentiation and access control in a real pervasive computing environment, distributed computer-assisted surgery [33].

Acknowledgments

This work was supported in part by Michigan Life Science Corridor under Grant number MEDC-459 and Wayne State University Faculty Research Grant.

References

[1] Advanced Dencryption Standard, URL http://csrc.nist.gov/CryptoToolkit/aes/

[2] Akamai Technologies Inc., Edgesuite services, URLhttp://www.akamai.com/html/en/sv/edgesuite_over.html

[3] E. Amir, S. McCanne, R. Katz, An active service framework and its application to real-time multimedia transcoding, in: Proceedings of the SIGCOMM'98, Vancouver, Canada, 1998.

[4] B. Baksi, R. Krishna, N. Vaidya, D. Pradhan, Improving performance of tcp over wireless networks, in: Proceedings of the 17th ICDCS, Baltimore, MA, 1997.

[5] A. Birrell, G. Nelson, S. Owicki, E. Wobber, Network objects, Software-Practice Experience 25 (S4) (1995) 87–130.

[6] A.T. Campbell, et al., A survey of programmable networks, ACM SIGCOMM Comput. Commun. Rev. 29 (2) (1999) 7–23.

[7] Y. Chow, W. Zhu, C. Wang, F.C. Lau, The state-on-demand execution for adaptive component-based mobile agent systems, in: Proceedings of ICPADS, Newport Beach, CA, 2004.

[8] L.P. Cox, C.D. Murray, B.D. Noble, Pastiche: making backup cheap and easy, in: Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation, Boston, MA, 2002.

[9] Data Dencryption Standard, URL http://www.itl.nist.gov/fipspubs/fip46-2.htm/

[10] Dicom standard, URL http://medical.nema.org

[11] W. Diffie, M. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory 22 (6) (1976) 644–654.

[12] F. Douglis, A. Iyengar, Application-specific delta-encoding via resemblance detection, in: Proceedings of the USENIX 2003 Annual Technical Conference, San Antonio, Texas, 2003.

[13] FIPS 180-1, Secure Hash Standard, US Department of Commerce /N.I.S.T., National Technical Information Service, Springfield, VA, 1995.

[14] A. Fox, S. Gribble, Y. Chawathe, E.A. Brewer, Adapting to network and client variation using infrastructural proxies: lessons and prespectives, IEEE Personal Comm. 5 (4) (1998) 10–19.

[15] A. Fox, S. Gribble, Y. Chawathe, E.A. Brewer, P. Gauthier, Cluster-based scalable network services, in: Proceedings of the 16th ACM Symposium on Operating Systems Principles, Saint-Malo, France, 1997.

[16] M. Freedman, E. Freudenthal, D. Mazires, Democratizing content publication with coral, in: Proceedings of the Sixth USENIX Operating Systems Design and Implementation, San Francisco, CA, 2004.

[17] X. Fu, W. Shi, A. Akkerman, V. Karamcheti, CANS: composable, adaptive network services infrastructure, in: Proceedings of the third USENIX Symposium on Internet Technologies and Systems (USITS'01), San Francisco, CA, 2001, pp. 135–146.

[18] A. Fuggetta, G.P. Picco, G. Vigna, Understanding code mobility, IEEE Trans. Software Eng. 24 (5) (1998).

[19] L. Gong, et al., Going beyond the sandbox: an overview of the new security architecture in the java development kit 1.2, in: Proceedings of the Usenix Symposium on Internet Technologies and Systems, Usenix Assn., Monterey, CA, 1997.

[20] S.D. Gribble, M. Welsh, E.A. Brewer, D. Culler, The multispace: an evolutionary platform for infrastructural services, in: Proceedings of the 1999 Usenix Annual Technical Conference, Monterey, CA, 1999.

[21] N.W. Group, Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms, URL http://rfc.net/rfc2001.html

[22] S. Gurun, C. Krintz, R. Wolski, Nwslite: a light-weight prediction utility for mobile devices, in: Proceedings of MobiSys'04, Boston, MA, 2004.

[23] Gzip tool. URL http://www.gzip.org

[24] D. Halls, Applying mobile code to distributed systems, Ph.D. Thesis, University of Cambridge, June 1997. URL http://www.crema.unimi.it/mirror/scheme/thesis/node6.html

[25] N.C. Hutchinson, L.L. Peterson, The *x*-kernel: an architecture for implementing network protocols, IEEE Trans. Software Eng. 17 (1) (1991) 64–76.

[26] A.D. Joseph, A.F. deLespinasse, J. Tauber, D. Gifford, M.F. Kaashoek, Rover: a toolkit for mobile information access, in: Proceedings of the 15th ACM Symposium on Operating Systems Principles, Cooper Mountain Resort, Colorado, 1995, pp. 156–171.

[27] A.D. Joseph, J.A. Tauber, M.F. Kasshoek, Mobile computing with the rover toolkit, IEEE Trans. Comput: Special Issue Mobile Comput. 46 (3) (1997) 337–352.

[28] E. Jul, H. Levy, N. Hutchinson, A. Black, Fine-grained mobility in the emerald system, ACM Trans. Comput. Systems 6 (1) (1988) 109–133.

[29] Kinoma player, URL http://www.kinoma.com/

[30] B. Krishnamurthy, J. Rexford, Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching and Traffic Measurement, Addison-Wesley, Inc, Reading, MA, 2001.

[31] LDAP (v3) revision (2004), URL http://www.ietf.org/ids.by.wg/ldapbis.html

[32] H. Lufei, W. Shi, L. Zamorano, On the effects of bandwidth reduction techniques in distributed applications, in: Proceedings of International Conference on Embedded and Ubiquitous Computing (EUC'04), Aizu, Japan, 2004.

[33] H. Lufei, W. Shi, L. Zamorano, Communication optimization for image transmission in computer-assisted surgery, in: Proceedings of 2004 Congress of Neurological Surgeons Annual Meeting (abstract), San Francisco, CA, 2004.

[34] A. Mallet, J. Chung, J. Smith, Operating system support for protocol boosters, in: Proceedings of HIPPARCH Workshop, Uppsala Sweden, 1997, pp. 10–15.

[35] U. Manber, Finding similar files in a large file system, in: Proceedings of the USENIX Winter 1994 Technical Conference, San Francisco, CA, 1994, pp. 1–10.

[36] R. Mohan, J.R. Simth, C. Li, Adapting multimedia internet content for universal access, IEEE Trans. Multimedia 1 (1) (1999) 104–114.

[37] A. Muthitacharoen, B. Chen, D. Mazières, A low-band width network file system, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP-18), Banff, Canada, 2001.

[38] P. Neumann (Ed.), Computer Related Risks, Addison Wesley, Reading, MA, 1995.

[39] B.D. Noble, Mobile data access, Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, May 1998. URL http://mobility.eecs.umich.edu/papers/diss.pdf

[40] B.D. Noble, et al., Agile application-aware adaptation for mobility, in: Proceedings of the 16th ACM Symposium on Operating Systems Principles SOSP-16, Saint-Malo, France, 1997.

[41] B.D. Noble, M. Price, M. Satyanarayanan, A programming interface for application-aware adaptation in mobile computing, in: Proceedings 2nd USENIX Symposium on Mobile and Location-Independent Computing, Ann Arbor, Michigan, 1995.

[42] P. Patel, D. Wetherall, J. Lepreau, A. Whitake, Tcp meets mobile code, in: Proceedings of the Ninth Workshop on Hot Topics in Operating Systems, Lihue, Hawaii, 2003.

[43] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, T. Stack, Upgrading transport protocols using untrusted mobile code, in: Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, New York, 2003, pp. 1–14.

[44] Planetlab, URL http://planet-lab.org/

[45] M.O. Rabin, Fingerprinting by random polynomials, Technical Report TR-15-81, Harvard Aiken Computation Laboratory, 1981.

[46] RC4 RFC3268, URL http://www.faqs.org/rfcs/rfc3268.html/

[47] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, J. Kubiatowicz, Pond: the oceanstore prototype, in: Proceedings of the Second USENIX Conference On File and Storage Technologies, San Francisco, CA, 2003, pp. 1–14.

[48] N.T. Spring, D. Wetherall, A protocol independent technique for eliminating redundant network traffic, in: Proceedings of ACM SIGCOMM'00, Stockholm, Sweden, 2000, pp. 87–95.

[49] P. Sudame, B. Badrinath, Transformer tunnels: a framework for providing route-specific adaptations, in: Proceedings of the USENIX Technical Conference, New Orleans, Louisiana, 1998.

[50] P. Sudame, B. Badrinath, On providing support for protocol adaptation in mobile wireless networks, Mobile Networks Appl. (MONET) 6 (1) (2001) 43–55.

[51] A. Tanenbaum, Modern Operating Systems, second Ed., Prentice-Hall, Englewood Cliffs, NJ, 2001.

[52] D. Tennenhouse, D. Wetherall, Towards an active network architecture, Comput. Comm. Rev. 26(2) 1996.

[53] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. Bressoud, A. Perrig, Opportunistic use of content addressable storage for distributed file systems, in: Proceedings of the USENIX 2003 Annual Technical Conference, San Antonio, Texas, 2003.

[54] P. Tridgell, P. Mackerras, The rsync algorithm, Technical Report TR-CS-96-05, Department of Computer Science, Australian National University, 1996.

[55] A. Vahdat, M. Dahlin, T. Anderson, A. Aggarwal, Active names: flexible location and transport of wide area resources, in: Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS'99), Boulder, Colorado, 1999.

[56] W3C Consortium, Simple object access protocol (SOAP) 1.1 (2000), URL http://www.w3.org/TR/SOAP/

[57] L. Wang, K. Park, R. Pang, V. Pai, L. Peterson, Reliability and security in the codeen content distribution network, in: Proceedings of USENIX Annual Conference, Boston, MA, 2004.

[58] D.J. Wethrall, J.V. Guttag, D.L. Tennenhouse, ANTS: a toolkit for building and dynamically deploying network protocols, in: Proceedings of Second IEEE OPENARCH, San Francisco, CA, 1998.

[59] Windows CE Operating Systems, URL http://www.microsoft.com/windowsce/

[60] M. Yarvis, A. Wang, A. Rudenko, P. Reiher, G.J. Popek, Conductor: distributed adaptation for complex networks, in: Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, Rio Rico, Arizona, 1999.

Weisong Shi is an Assistant Professor of Computer Science at Wayne State University. He received his B.S. from Xidian University in 1995, and Ph.D. degree from the Chinese Academy of Sciences in 2000, both in Computer Engineering. His current research focuses on dynamic Web content delivery, trusted resource sharing in peer-to-peer systems, mobile computing, and wireless sensor networks. Dr. Shi has published more than 40 peer-reviewed journal and conference papers in these areas. He is the author of the book ”Performance Optimization of Software Distributed Shared Memory Systems” (High Education Press, 2004). He has also served on technical program committees of several international conferences, including the chair of poster track of WWW 2005. He is a recipient of Microsoft Fellowship in 1999, the President outstanding award of the Chinese Academy of Sciences in 2000, one of 100 outstanding Ph.D. dissertations (China) in 2002, ”Faculty Research Award” of Wayne State University in 2004 and 2005, the ”Best Paper Award” of ICWE'04 and IPDPS'05. He is a member of ACM, USENIX, and IEEE.

Hanping Lufei is a Ph.D. student of computer science at Wayne State University. His current research focuses on QoS, systems security, access control, and trust management in mobile computing environment. He is also interested in computing enhancement for handheld device and resource management in distributed systems. He received his Bachelor degree in 1998 and Masters degree in 2001 from Huazhong University of Science and Technology (HUST) in China and the University of Toledo in USA, both in Electrical Engineering.