NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SIMULATION MODELING AND ANALYSIS OF DEVICE-AWARE NETWORK ARCHITECTURES

by

Jin Hou, KOH

December 2004

Thesis Advisor:	Gurminder Singh
Thesis Co-Advisor:	Su Wen

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Simulation Modeling and Analysis of Device-Aware Network Architectures			**5. FUNDING NUMBERS**
6. AUTHOR(S) Jin Hou Koh			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

As the popularity of Internet soars, the content on the Internet is increasingly accessed by mobile devices that are usually small in form factor and limited in resources, in terms of processing capability, bandwidth and battery power. With the changing environment, content providers must serve a large number of access devices with different profiles, while the users have access to a large number of services with different content types. A key challenge in such an environment is how to enable the best possible fit between content and capabilities of a specific access device type.

The goal of this thesis research is to explore on the concept of a device-aware network (DAN) that can provide the infrastructure support for device-content compatibility matching to avoid the unnecessary wastage of network and device resources that happens in current device-ignorant networks. A more efficient architecture is proposed which encapsulates device profile information in transmitting packets and incorporates content repurposing functionality in existing network entities, such as routers along the data path. Simulation models are developed to statistically evaluate the performance of the proposed architecture in comparison to existing content repurposing frameworks. The results demonstrated the feasibility and suitability of the architecture, with improvement in network bandwidth conservation.

14. SUBJECT TERMS Communications, Device-Aware Network, Content Repurposing, Device Profiling, Simulation, OPNET, OMNeT++.			**15. NUMBER OF PAGES** 103
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SIMULATION MODELING AND ANALYSIS OF DEVICE-AWARE NETWORK ARCHITECTURES

Jin Hou, KOH
Civilian, Singapore Defence Science and Technology Agency
B.Eng. (First Class Honors), National University of Singapore, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2004

Author: Jin Hou, KOH

Approved by: Gurminder Singh
Thesis Advisor

Su Wen
Thesis Co-Advisor

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

As the popularity of Internet soars, the content on the Internet is increasingly accessed by mobile devices that are usually small in form factor and limited in resources, in terms of processing capability, bandwidth and battery power. With the changing environment, content providers must serve a large number of access devices with different profiles. A key challenge in such an environment is how to enable the best possible fit between content and capabilities of a specific access device type.

The goal of this thesis research is to explore the concept of a device-aware network (DAN). A device aware network can provide the infrastructure support for device-content compatibility matching to avoid the unnecessary wastage of network and device resources that happens in the current device-ignorant networks. A more efficient architecture is proposed which encapsulates device profile information in transmitting packets and incorporates content repurposing functionality in existing network entities, such as routers along the data path. Simulation models are developed to statistically evaluate the performance of the proposed architecture in comparison to existing content repurposing frameworks. Our results demonstrate the feasibility and suitability of the architecture, with improvement in network bandwidth conservation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to thank Professors Gurminder Singh and Su Wen for their support and guidance throughout the course of his thesis work. Their keen interest and knowledge in the subject area and technical expertise have been a great source of assistance. The author is extremely grateful for his wife Meng Geah, LING who has constantly provided him with her support, unselfish love, and understanding throughout the research.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Until recently, there were a few select ways of accessing the Internet, mainly through desktop PCs and workstations. However, the current trend is to access Internet content and applications anytime, anywhere and on any device. More and more wireless and mobile users with different terminals use the various services available in the Internet. In fact, more than 600 wireless and mobile device profiles are documented for accessing on-line information [W3C, 2004]. The range of access devices includes mobile phones, Personal Digital Assistants (PDAs), desktops, laptops, wearable PCs, and set-top boxes. With military transformation towards network-centric warfare, surveillance devices, sensors, Unmanned Aerial Vehicles (UAVs), launching and targeting platforms, and missiles are likely to be networked, which will bring the number of access devices to an even larger figure.

With the changing environment, the service and content providers must serve a large number of access devices with different profiles. Figure 1 exemplifies this environment. The key challenge in such an environment is how to enable the best possible fit between the content to be delivered and the capabilities of the specific access device.

Most of today's content and applications are designed for desktop PCs and workstations with large color screens, ample CPU power and broadband Internet access. Unfortunately, most of the other access devices have different profiles, particularly wireless and mobile devices, which are predominantly smaller in form factor and have limited resources – they differ in network connectivity, storage, memory, battery power, processing power, display, and format handling capabilities. To give an example, desktop PCs have a screen resolution of at least 1024 x 768 pixels, while PDAs have displays with a maximum of 240 x 320 pixels, and the screens of mobile phones are even smaller at 176 x 208 pixels or less. If a large image, which displays well on a PC, is delivered to small handheld device, it may be too overwhelming for the screen of the handheld device and the user of the device. In addition, the handheld device may lack the memory and processing power to handle the image. Receiving such an image will only result in the

device discarding it, thus resulting in wastage CPU processing, network bandwidth, device battery and time. For these reasons, device awareness becomes necessary to optimize user experience and minimize wastage of resources, so that "usable" content delivery can be achieved.

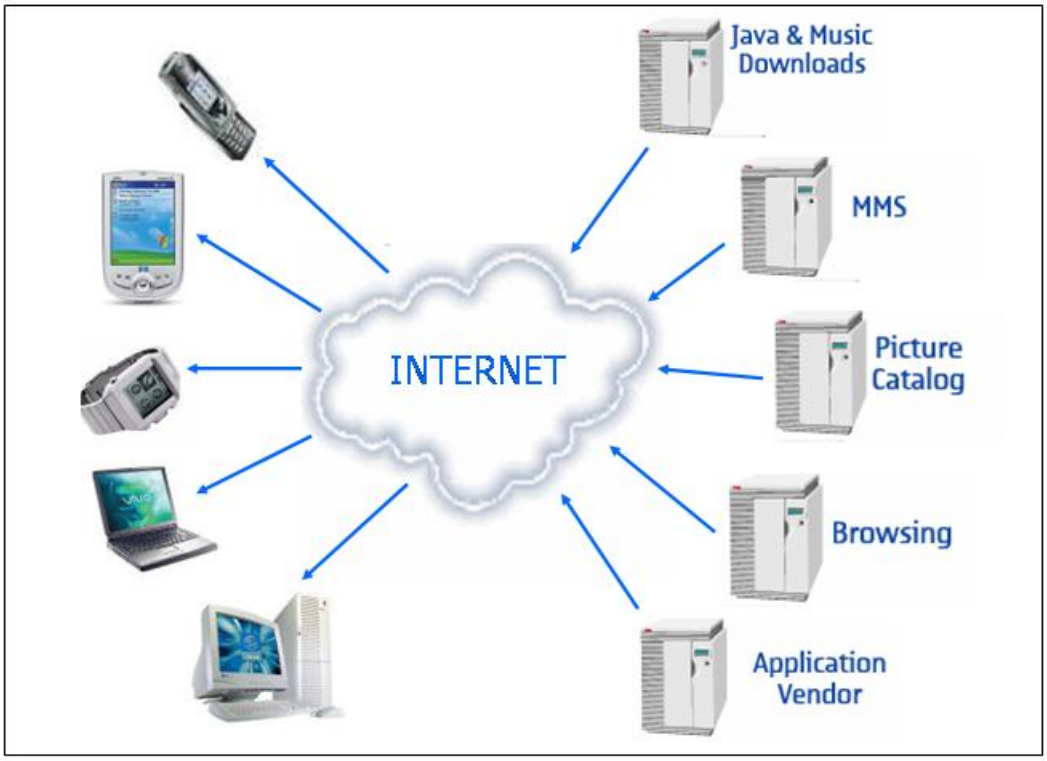


Figure 1. Challenge of having different access devices accessing to different content type (After Ref [Nokia, 2003])

Today's networks are completely unaware of the capabilities of their end-points. Being dumb transport pipes, they cannot optimize the traffic that flows through them and adapt the content to match the capabilities and requirements of their end-points. The goal of this thesis research is to explore the concept of a device-aware network (DAN) that can provide the infrastructure support for device-content compatibility matching. Resources unnecessary expended handling "unusable" content would have been better utilized for appropriately formatted content, especially in time-sensitive networks.

With DAN, the benefits will include, not limited to the following:

Better user experience. DAN provides the means to deliver the most suitable content for a specific device type. It avoids the situation whereby a user faces frustration when an "unusable" content is delivered to the device.

Minimize wastage of critical resources. DAN is particularly useful in a wireless and mobile environment, where devices with small form factor are more prevalent and resources such as time, network bandwidth and battery power are limited and scarce.

Transparency for content providers. Leveraging on DAN, content providers do not need to be bothered about keeping multiple copies of content and handcrafting content for different device type, which is expensive to implement and a management nightmare.

A. ORGANIZATION

This chapter provides an introduction to the problem and the motivation for the research. The approach for the research is to explore on various architectural designs suitable to support device-aware networking, by examining relevant technologies and implementations. Simulation models of the proposed architectural approaches are developed to investigate design considerations of device-aware networks. The models will serve to provide guidance on suitability of different architectural designs. Chapter II provides a review of current technologies and implementations relevant to device-aware networking. Chapter III introduces the discrete event simulation tools OPNET and OMNET++ that are used to model the proposed system designs for device-aware networks. Chapter IV proposes possible designs for the architectural framework to enable device awareness, while Chapter V documents the development of the device-aware network simulation models for performance evaluation, and discusses the analysis of the simulation results. Chapter VI discusses recommendations of the proposed model for device-aware network based on the simulation results, and also presents the conclusions that can be drawn from this research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. REVIEW OF RELATED WORK

The Internet today relies largely on the Internet Protocol (IP) for communications among interconnected computer systems. The delivery mechanism is a best-effort service, and the implementation is a simple one – payload is encapsulated in IP datagrams with IP headers, and intermediate entities such as network routers make use of the fixed-length addresses in the IP headers to transmit the datagrams towards their destinations [RFC791, 1981]. There are no mechanisms to augment end-to-end data reliability, flow control or sequencing. These services are usually implemented as host-to-host protocols.

To enable device-aware networking, the network architecture will require some modifications to provide additional services than best-effort delivery in order to achieve the objective of optimizing traffic to match the capabilities and requirements of end-points. DAN needs to provide at least three main services – the repurposing of the content to match the capabilities and requirements of the end devices; the sharing and delivery of device profile and capability information on the network; and capability negotiation.

This chapter provides a review of the current technologies and implementations relating the three services which DAN will need to provide, namely content repurposing, device profiling and content negotiation. Section A examines the three approaches currently practiced – client-based, server-based and proxy-based, for content repurposing and discusses the advantages and limitations of the different approaches. Section B presents device profiling techniques, while Section C discusses capability negotiation process to determine compatibility of content to end device.

A. CONTENT REPURPOSING

Content repurposing is described as the process of selection, generation or modification of content (text, images, audio and video) to suit to the user's computing environment and usage context [Singh, 2004], [MediaLab, 2004]. It can be applied to transformation within media types, for example reducing the image size or transforming

from high-fidelity color JPEG to low-fidelity GIF format; across media types, for example speech to text or video item to image set; or to both of them [Canali et al, 2003].

To get a clearer understanding of content repurposing, take an example of a normal web page accessed by a PC connected to the Internet via broadband. The user will see the original web page with all text, images and video, without repurposing. However, when the same web page is accessed using a PDA, the user will not see the same content. Images would be rescaled and compressed, text would be summarized into a single paragraph, and if there is video in the original content, it would be delivered as a set of images. Figure 2 illustrates this concept of content repurposing, where the original results of a Google search is repurposed (reduction in the amount of text and images used) to suit a Nokia 6230 WAP-enabled mobile phone and an HP iPAQ Pocket PC handheld.

Figure 2. Results of a Google search repurposed for WAP-enabled mobile phone and PDA (From Ref [MediaLab, 2004])

There are three widely-used approaches for content repurposing depending on the entities that perform the repurposing process. The content can be repurposed at the client end, on the server, or in an intermediate entity called a proxy. The following sections will discuss in details the different approaches for content repurposing.

1. Client-based Repurposing

In the client-based approach, the client device performs the required repurposing. The content is transmitted in its original form from the server to the client, as depicted in Figure 3.

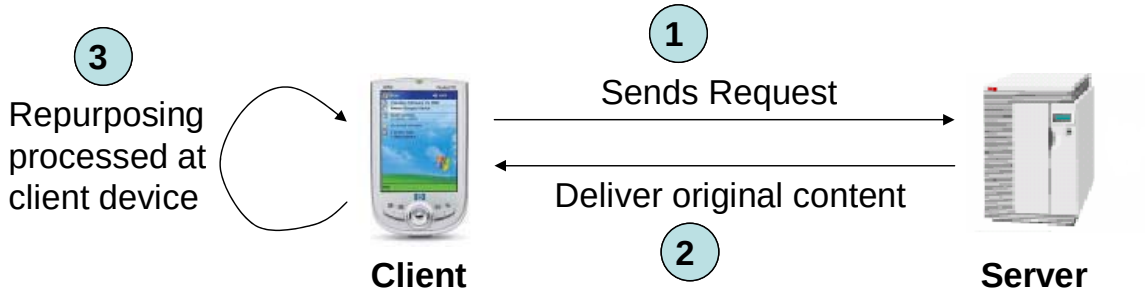


Figure 3. Client-based approach

In this approach, existing communication protocols do not need to be changed since the client does not need to communicate any information about its capabilities and requirements to the server, and content providers do not need to keep multiple versions of the same content to meet the requirements of different device types. However, content repurposing process generally involves computationally intensive operations. Adopting client-based approach will be very time consuming for wireless and mobile devices which have limited battery power, processing power and low-bandwidth connection. Furthermore, client-based approach does not make sense from the network optimization perspective since the content traverses across the network in its original size only to have parts of information discarded or reduced subsequently at the client end.

Opera Software is an example that employs client-based content repurposing. It uses a proprietary “Small-Screen Rendering” technology in its web browser for mobile wireless devices that intelligently reformats today’s web sites to fit inside smaller screen width, eliminating the need for horizontal scrolling [Opera, 2004].

2. Server-based Repurposing

In the server-based approach, the server adapts the content to match the requesting device profile and specifications. This is illustrated in Figure 4.

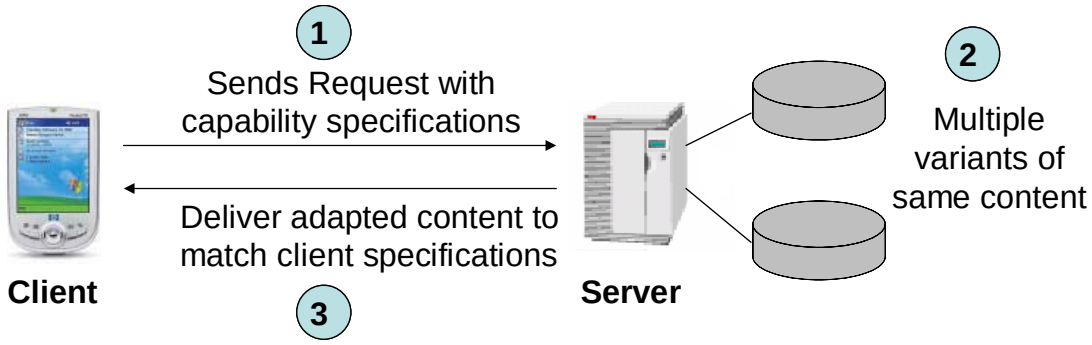


Figure 4. Server-based approach

There are many techniques for achieving server-based repurposing. Traditionally, content authors handcraft the same content for different device specifications, and the multiple variants are stored on the server. Appropriate version is then selected to match the client specifications. Adopting server-based repurposing, transmission times are reduced and network bandwidth usage is optimized since it involves delivery of already adapted content. Furthermore, already adapted content will be less taxing in terms of battery power and processing power for wireless and mobile devices. However, from the perspective of the content provider, it is expensive to maintain multiple variants of the same content, especially if the size of the clients requiring some form of repurposing is unknown. If real-time repurposing is performed, extra computational load will be inevitably added to the server.

IBM WebSphere Transcoding Publisher is an example of a server-based repurposing technology that dynamically translates web content and applications into multiple markup languages and optimizes it for delivery to mobile devices, such as mobile phones and handheld computers [WebSphere, 2004]. Such technology for adapting content for many devices and languages, at least eliminates the need to store multiple variants on the server.

3. Proxy-based Repurposing

In the proxy-based approach, a proxy server, located in an intermediate position along the communication path between the client device and the content server, will

analyze and perform the required repurposing on the requested content, before delivering the adapted content to the client device, as shown in Figure 5.

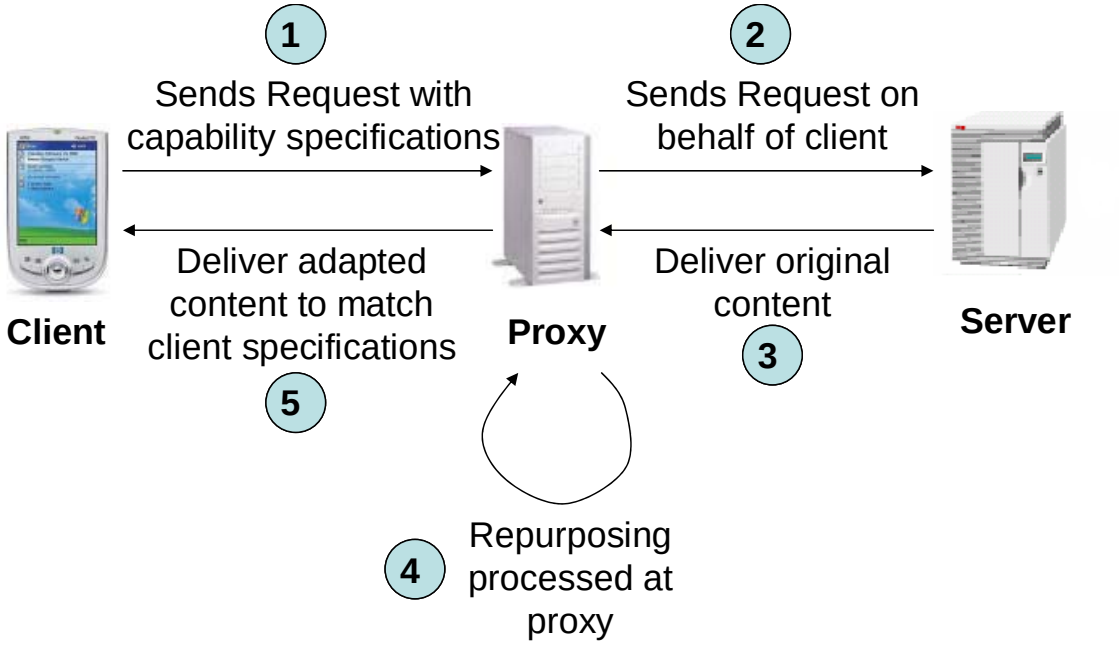


Figure 5. Proxy-based approach

In this approach, neither the client nor the server needs to be modified for content repurposing. Having offloaded the computationally intensive repurposing process to the intermediate proxy, computational load on the server will be reduced. However, a potential issue lies in the location of the intermediate proxy with respect to the content servers; if the proxy is far away from the content server (network connectivity sense), the original content will still need to traverse over long distance before adaptation at the proxy, and that is not optimizing network bandwidth.

AvantGo service is an excellent example of a system implemented using the proxy-based approach [AvantGo, 2004]. In a nutshell, a user subscribes to an AvantGo channel, which is a mobile website with personalized and reformatted content for PDAs and smartphones. The entire reformatting process is shown in Figures 6, 7, and 8. When a user syncs the mobile device, a connection will be established to the AvantGo sync server. The AvantGo server, after looking up what channels the user is subscribed to, will download those web pages from the relevant sites on the Internet. The AvantGo sync

server will pre-process these pages, which include shrinking images too large for the mobile device's screen, discarding pieces that cannot be used by the AvantGo Client (such as Java applets), and compressing the rest of the HTML. Once that is done, the compressed pages will be uploaded to the mobile device.

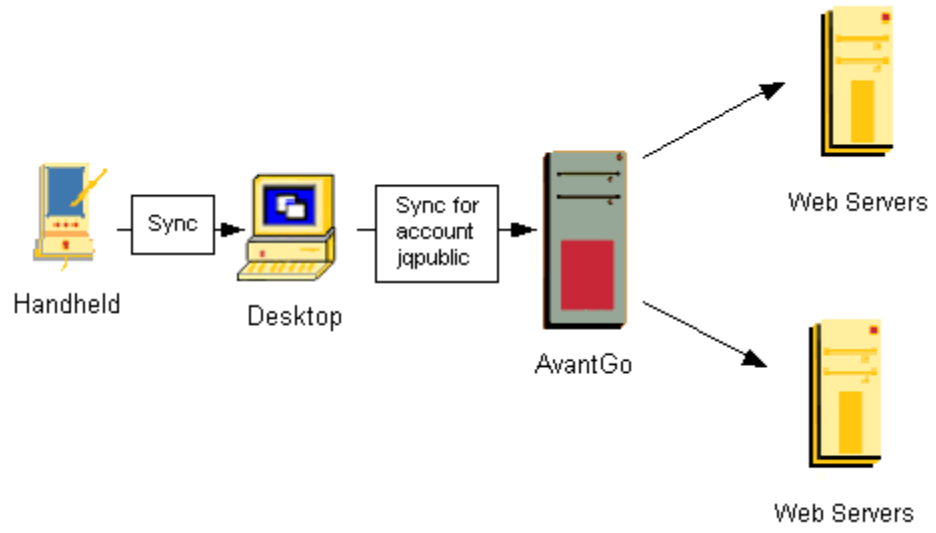


Figure 6. Sync request from mobile device (From Ref [AvantGo, 2004])

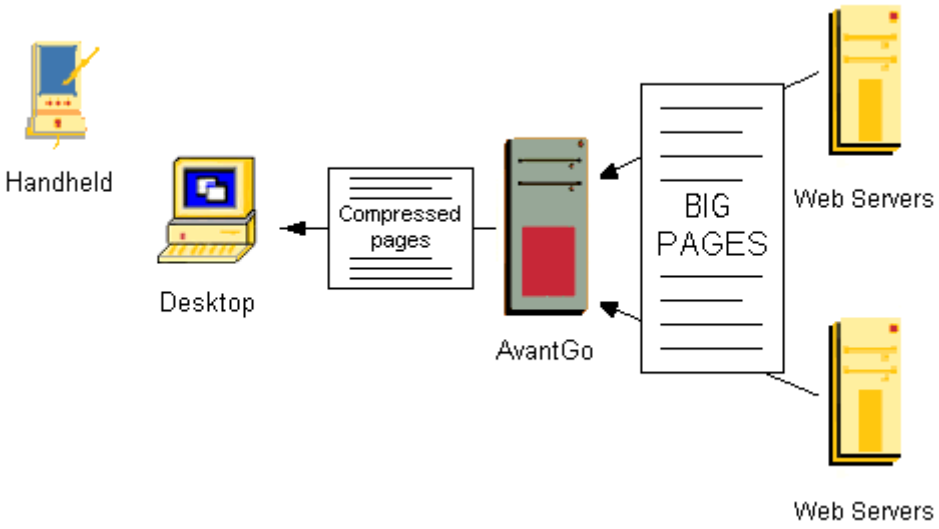


Figure 7. Sync response to desktop, after repurposing performed by AvantGo sync server - an intermediate proxy server (From Ref [AvantGo, 2004])

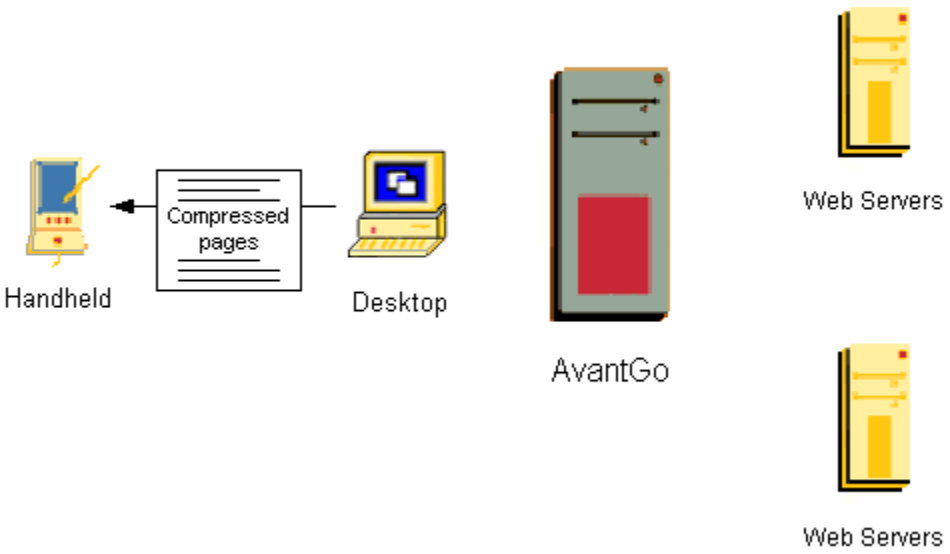


Figure 8. Adapted content relayed to mobile device (From Ref [AvantGo, 2004])

The process described is for subscribers who access the service by syncing their mobile devices via laptops or desktops – the "offline" mode. The service also supports real-time, wireless mode using 802.11b WiFi, Bluetooth connections, as well as cellular networks.

B. DEVICE PROFILING

The previous section described various approaches for content repurposing. Before the content servers and proxies can modify the content for a given device, they need to know what kind of device is making the requests. The methods for devices to communicate their capabilities and preferences to the servers will be discussed in this section.

1. Composite Capability / Preference Profiles (CC/PP)

As the number and variety of devices connected to the Internet grows, there is a corresponding increase in the need to deliver content tailored for the different devices. The Composite Capability / Preference Profiles (CC/PP), which becomes a World Wide Web Consortium (W3C) recommendation on 15 January 2004, is a general purpose profile format that describes device capabilities and user preference that can be used to

guide the adaptation of content presented to that device [CC/PP, 2004]. The strength of CC/PP lies in its flexibility. CC/PP is based on RDF, the Resource Description Framework – a general purpose metadata description language – that allows the creation of whole vocabularies, making the expression of device and agent capability, as well as user preference, infinitely extensible [WASP, 2004].

Using CC/PP, producers of devices and user agents can easily define precise profiles for their products, while content servers and proxies can use these profiles to repurpose the content they serve to the requirements of the devices.

Figure 9 shows a graph that provides an example of how CC/PP can be used to describe device capability and user preference.

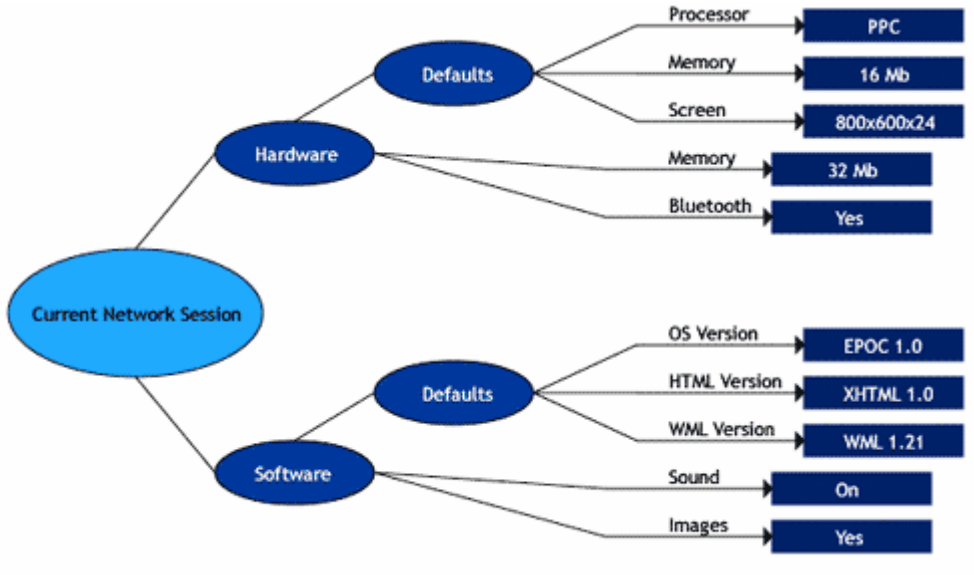


Figure 9. A CC/PP graph explanation (After Ref [WASP, 2004])

2. User Agent Profile (UAProf)

User Agent Profile (UAProf) specification, developed by Open Mobile Alliance (OMA), uses CC/PP model to define a framework for describing and transmitting "Capability and Preference Information" (CPI) about WAP (Wireless Application Protocol)-enabled devices [WAG, 2001]. UAProf is an XML-format document which is published on a public repository server, and it contains device capability information such as hardware characteristics (screen size, color capabilities, image capabilities, manufacturer, etc), software characteristics (operating system vendor and version, list of

audio and video encoders, etc.), application/user preferences (browser manufacturer and version, markup languages and versions supported, scripting languages supported, etc.), WAP characteristics (WML script libraries, WAP version, WML deck size, etc.), and network characteristics (bearer characteristics such as latency and reliability, etc.).

The architecture by which UAProf is transported between the mobile device, WAP Gateway and content server is illustrated in Figure 10. The WAP Gateway supports UAProf header forwarding. Though UAProf XML-files are comprehensive, they tend to be large in size as well. Take for example; the UAProf XML-file for Nokia 6650 phone, which is shown in Appendix A, is 12 KB. For this reason, device vendors usually have their own device profile repository where content servers can download device profiles as XML documents. Mobile devices will provide the profile document URL in their request session header to the content server, as shown in Figure 10.

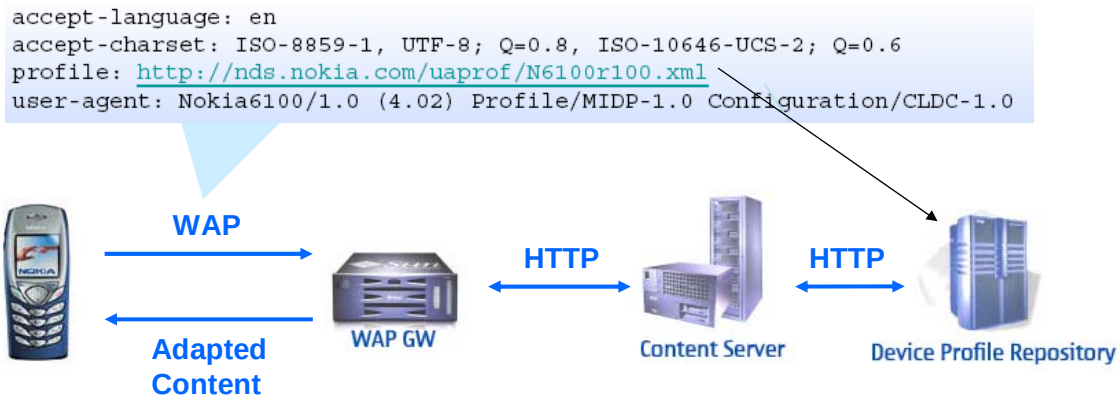


Figure 10. Architecture using UAProf (After Ref [Nokia, 2003])

C. CAPABILITY NEGOTIATION

1. Session Initiation Protocol (SIP) and Session Description Protocol (SDP)

The Session Initiation Protocol (SIP), as defined in [RFC2543, 1999] is a lightweight application-layer control protocol that can establish, modify and terminate multimedia sessions or calls. It is an out-of-band protocol that is used to initiate sessions between end systems. It is relevant to the design of DAN because of its potential as a means to communicate and exchange device capabilities and user preferences between

end systems. Another important functionality of SIP, which is of relevance to the design of DAN, is the determination of media and media parameters to be used in multimedia sessions between participants – capability negotiation. SIP uses Session Description Protocol (SDP) specified in [RFC2327, 1998] as a data format to describe and convey multimedia sessions.

To understand the essence of SIP, it is best to look at a simple SIP call, as illustrated in Figure 11 [Kurose, 2003].

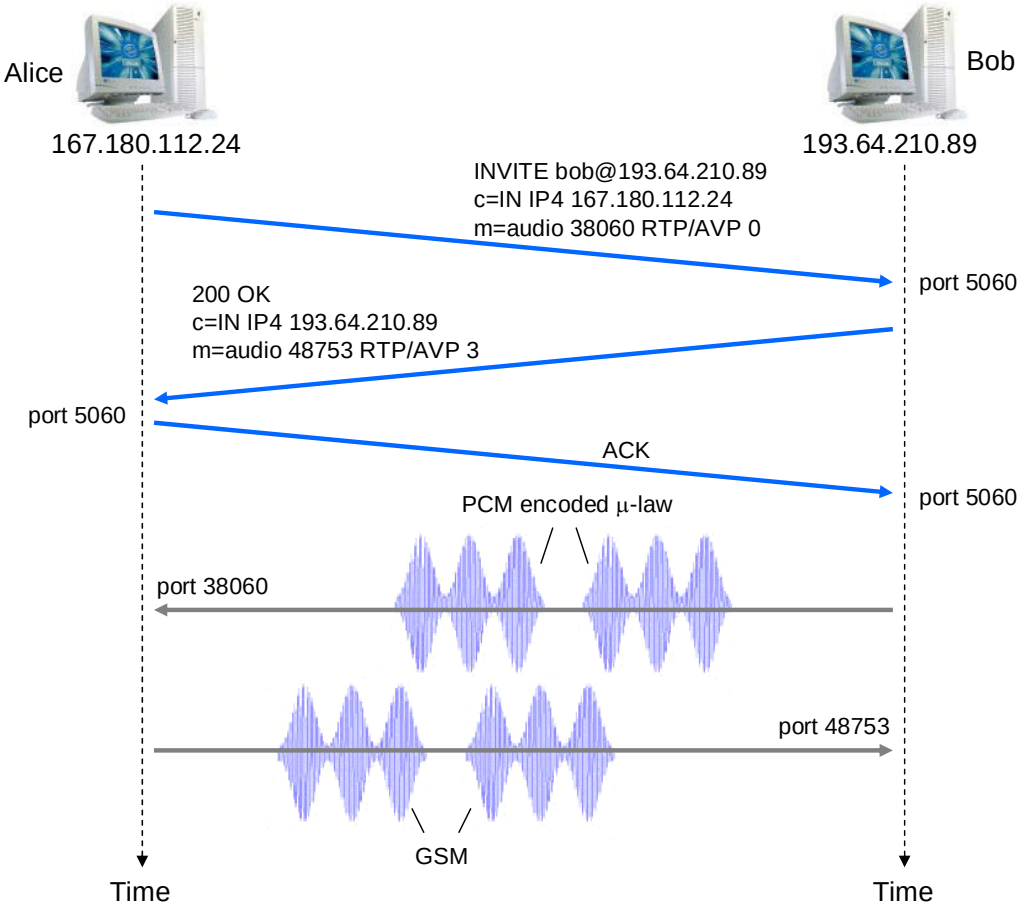


Figure 11. A simple SIP session (After Ref [Kurose, 2003])

As shown in Figure 11, the SIP session begins when Alice sends Bob an INVITE message on SIP well-known port of 5060. The message contains enough information to establish a multimedia session between participants. This information includes media

capabilities that Alice can receive and the transport address where Alice expects Bob to send the media data. In this example, Alice is expecting PCM μ-law audio (AVP 0) encapsulated in RTP, at IP address 167.180.112.24 on port 38060. Bob sends back an OK response, indicating that he is accepting the request, as well as information describing his desired encoding and packetization (GSM audio encapsulated in RTP – RTP/AVP 3), and port number (48753) to receive the data. Once the initiation session is completed, the multimedia session will follow with each party receiving media according to the requested encoding and transported on the requested port number.

There also exists an ongoing initiative called SDPng (SDP Next Generation) which extends SDP to include capability negotiation [SDPng, 2004]. SDP is designed to provide description of a session parameters, but it is inadequate to describe all capabilities of a system and possibly provide a choice between a number of alternatives. This is the objective of SDPng. SDPng is an application-independent framework that defines the description syntax for both potential and actual capabilities, and the processing rules that are applied in the capability negotiation process to generate an inter-working capability from two or more usable potential capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

III. OVERVIEW OF NETWORK SIMULATION TOOLS – OPNET AND OMNET++

A. OPNET

OPNET (Optimized Network Engineering Tools) was originally developed at MIT and introduced in 1987 as the first commercial network simulator. OPNET provides a comprehensive development environment supporting the modeling of communication networks and distributed systems [OPNET, 2004]. Some typical applications of OPNET include, but not limited to the following:

- Performance modeling of standards-based Local Area Networks (LAN) and Wide Area Network (WAN).
- Planning of large internetworks.
- Research and development in communication architectures and protocols.
- Mobile packet radio networks.

OPNET categories the simulation software into several products to better provide users with their specific needs. Some of the products and the specific domains for they are designed, are listed below:

- *IT Guru*. OPNET IT Guru enables enterprise users in operations, planning and application development to be far more effective in confronting the challenges in cost-effective management of networks and applications, as their reliance on IT infrastructure continues to increase.
- *ACE*. OPNET Application Characterization Environment (ACE) module enables IT Guru users identify the root-cause of end-to-end application performance problems.
- *Modeler*. OPNET Modeler is a powerful modeling and simulation platform, and provides a network development environment, essential for design and analysis of networks, network equipment, and communication protocols in the research and development domain.
- *Wireless Module*. OPNET Wireless Module extends the functionality of OPNET Modeler with modeling, simulation, and analysis of wireless networks.

For the simulations in this thesis, OPNET Modeler has been used with Wireless Module. A brief overview of OPNET Modeler is provided in the following few sections in this chapter.

1. Modeling Architecture

OPNET modeling architecture consists of hierarchical models, paralleling the structure of actual communications networks. Each hierarchical level of an OPNET model is referred to as a modeling domain. The top-level modeling domain is the network domain. The network domain contains node domain, which in turn contains module domain. Lastly, lowest-level modeling domain, the process domain is nested within the module domain. The relationship between the domains is illustrated in Figure 12.

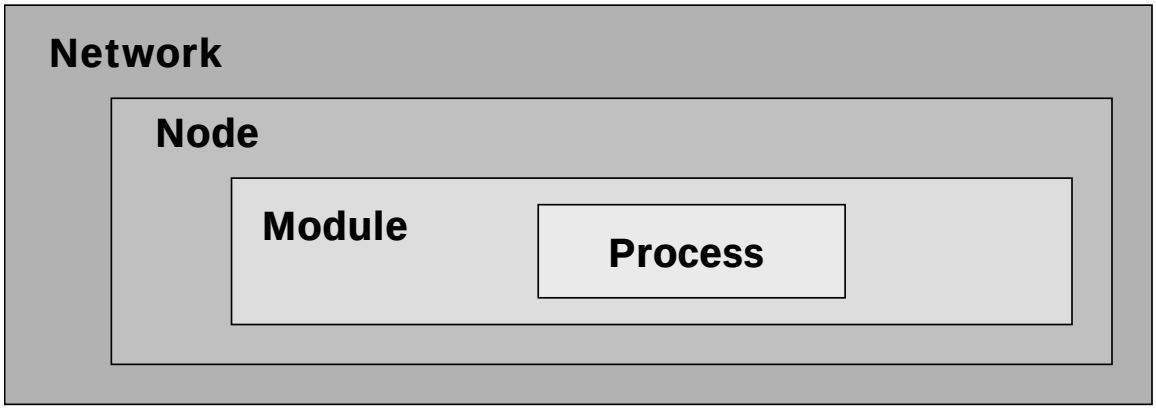


Figure 12. Hierarchical levels of OPNET model (After Ref [OPNET, 2004])

The network domain is used to define the topology of a communications network. The communicating entities within a network domain are called nodes, and a network domain can support any number of nodes. These nodes are interconnected via communication links such as point-to-point and bus links.

The node domain provides for the modeling of communication devices, such as workstations, switches, and routers. Each node domain is expressed in terms of smaller building blocks called modules. Modules are information sources, sinks, processors and queues. Connection such as packet stream will allow data flow between modules. The tasks performed by each module are called processes, which are sets of instructions, much like executing a software program.

The process domain is expressed in a language called Proto-C, which is an OPNET variant on the C/C++ language, specifically designed to support developing protocols and algorithms. Proto-C is a compiled language combining graphical state-transition-diagrams (STDs), embedded C/C++ language data items and statements, and a

library of Kernel Procedures that provide commonly needed functionality for modeling communications and information processing systems. Process models are represented by finite state machines (FSMs), which define a set of primary states that the process can enter, and for each state, the conditions that would cause the process to transit to another state. The conditions, needed for a particular change in state to occur and the associated destination state are called a transition. Operations and actions performed in each state or for a transition are described in embedded C/C++ code blocks. In addition, OPNET also provides an extensive library of over 300 Kernel Procedures that can be invoked within the process models by the simulation kernel to do commonly needed operations and actions.

The modeling function addressed by each domain is summarized in Table 1, and the graphical editors for the network, node and process domains are shown in Figure 13.

Domain	Modeling Function
Network	Network topology described in terms of subnetworks, nodes, links, and geographical context.
Node	Node internal architecture described in terms of functional elements and data flow between them.
Module	Modules include information source, sink, processor and queue. Some modules have pre-defined behavior while others are programmable via process model.
Process	Behavior of processes (protocols, algorithms and applications) specified using finite state machines and extended high-level languages (C or C++ programming language).

Table 1. OPNET modeling domains (After Ref [OPNET, 2004])

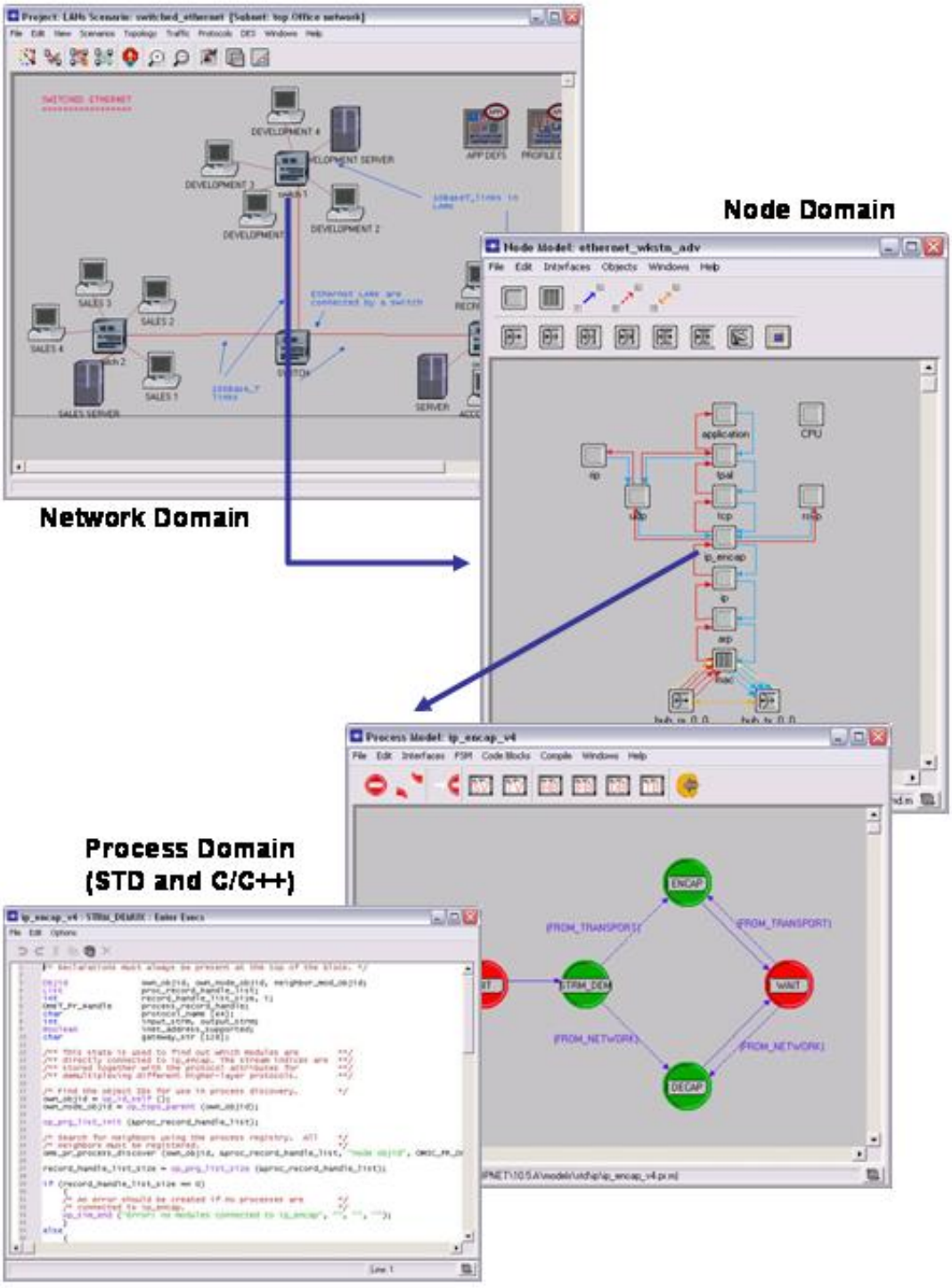


Figure 13. Graphical editors for network, node and process models

2. Modeling Application Traffic

After the network topology, the next step is to provide application traffic for exchange of information within the network. OPNET provides standard client-server application models – FTP, Email, Remote Login, Video Conferencing, Database access, HTTP, Print service, Voice communications, and Custom application. Of particular interest is Custom application, which is a user-definable multi-tier application and is used extensively for the simulation work in the course of this research. It will be elaborated in this section.

a. Terminology in Custom Application

In configuring custom application, there are a few definitions that need to be clarified. They are defined as follows, and explained pictorially in Figure 14:

- *Task*: A basic unit of user activity within the context of an application. Examples of a task include reading an email message; obtaining a record from a database system; and performing a file transfer.
- *Phase*: An interval of related activity that is contained within a task. Examples of a phase include data transfer phase and processing phase.
- *Step*: A phase is made up of a number of steps. Take for example; obtaining a single record from a database server involves at least 2 steps. The first step is to send a request to the database server, and the second step is to receive the corresponding response from the database server.

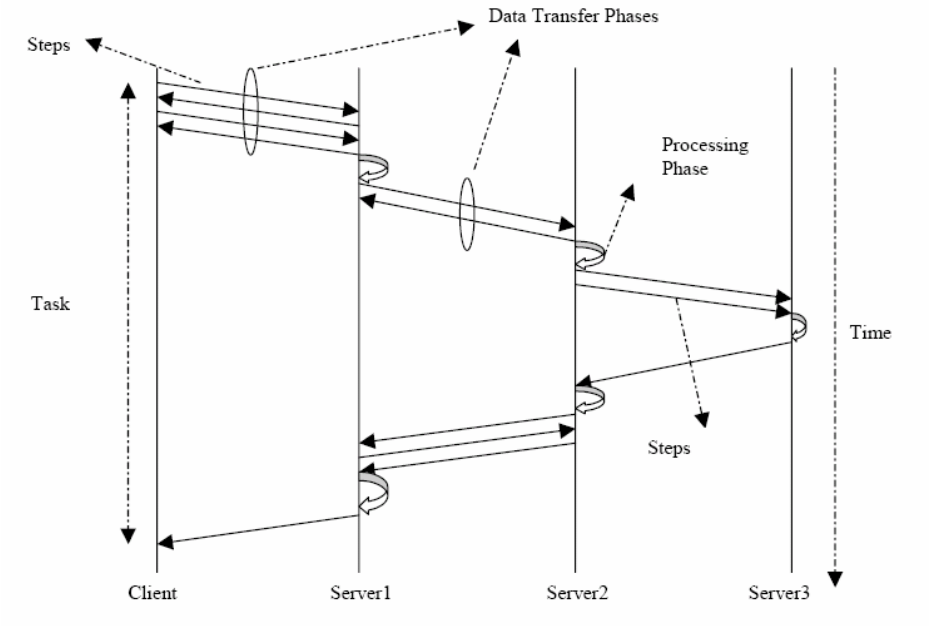


Figure 14. Custom application modeling terminology (From Ref [OPNET, 2004])

b. Configuring Tasks and Phases

Tasks and phases are building blocks of custom application, and they are defined in Task Definition Object, as shown in Figure 15. Tasks are configured in the task specification table, and there can be many tasks depending on the defined application. In the example, only one task is shown. Within each task, there are many phases configured. For the example shown in Figure 15, the sample configuration table shows a task with 6 phases. Transactions #1, #3, #5 and #6 are data transfer phases with defined entities as sources and destinations. Transactions #2 and #4 are processing phases as they have Destination attribute set to Not Applicable.

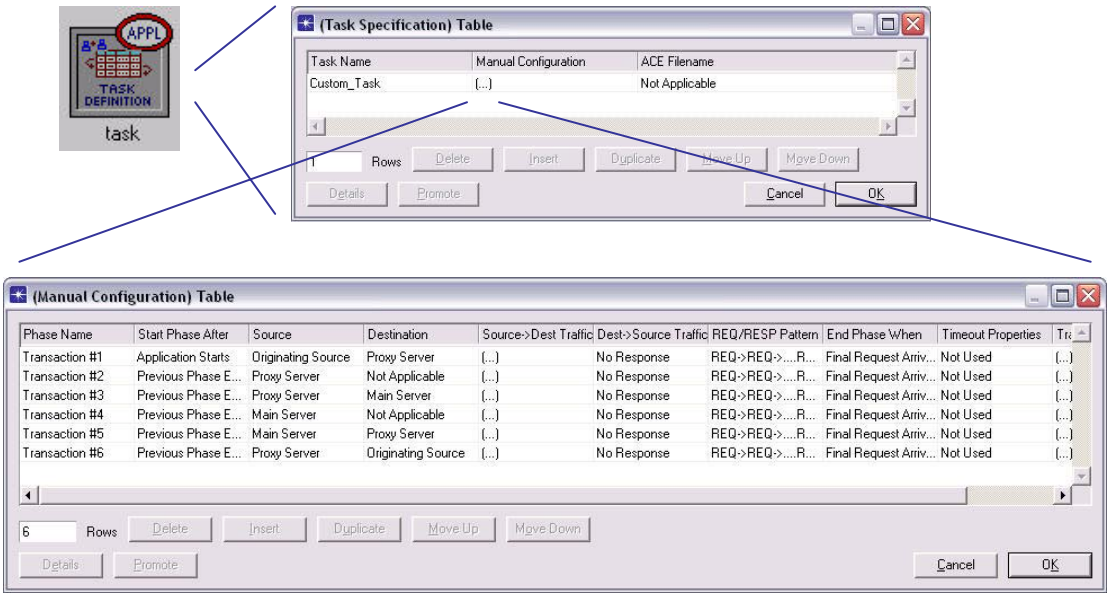


Figure 15. Sample configuration of tasks and phases

c. Configuring Applications and Profiles

Once the tasks and phases are defined in Task Definition Object, they can be used to build custom applications in Application Definition Object. The application definitions can then be used in profiles that are deployed to client workstations using the application. These profiles are defined in Profile Definition Object. Profiles are constructed to describe activity patterns of a user or a group of users in terms of the applications used over a period of time. The hierarchical structure of building custom application in OPNET is summarized in Figure 16. A sample configuration of application

definition in Application Definition Object is illustrated in Figure 17, while a sample configuration of profile definition in Profile Definition Object is illustrated in Figure 18.

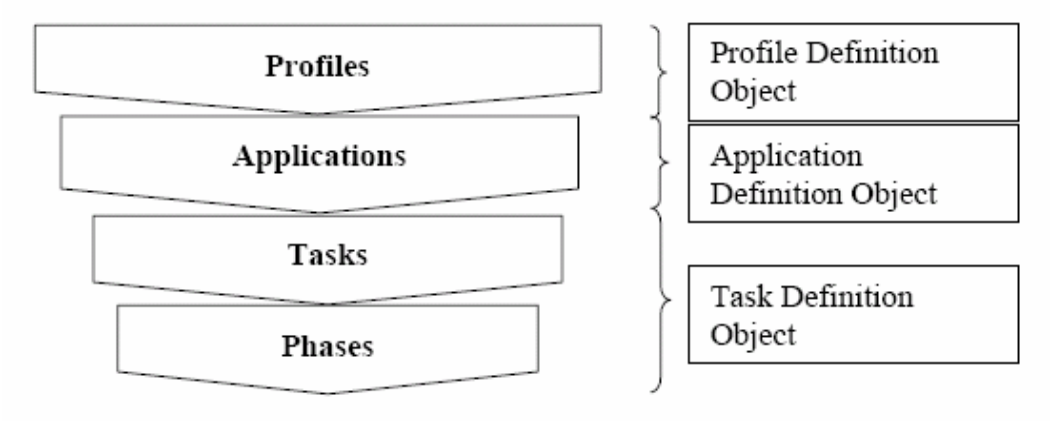


Figure 16. Hierarchical structure of building application model (From Ref [OPNET, 2004])

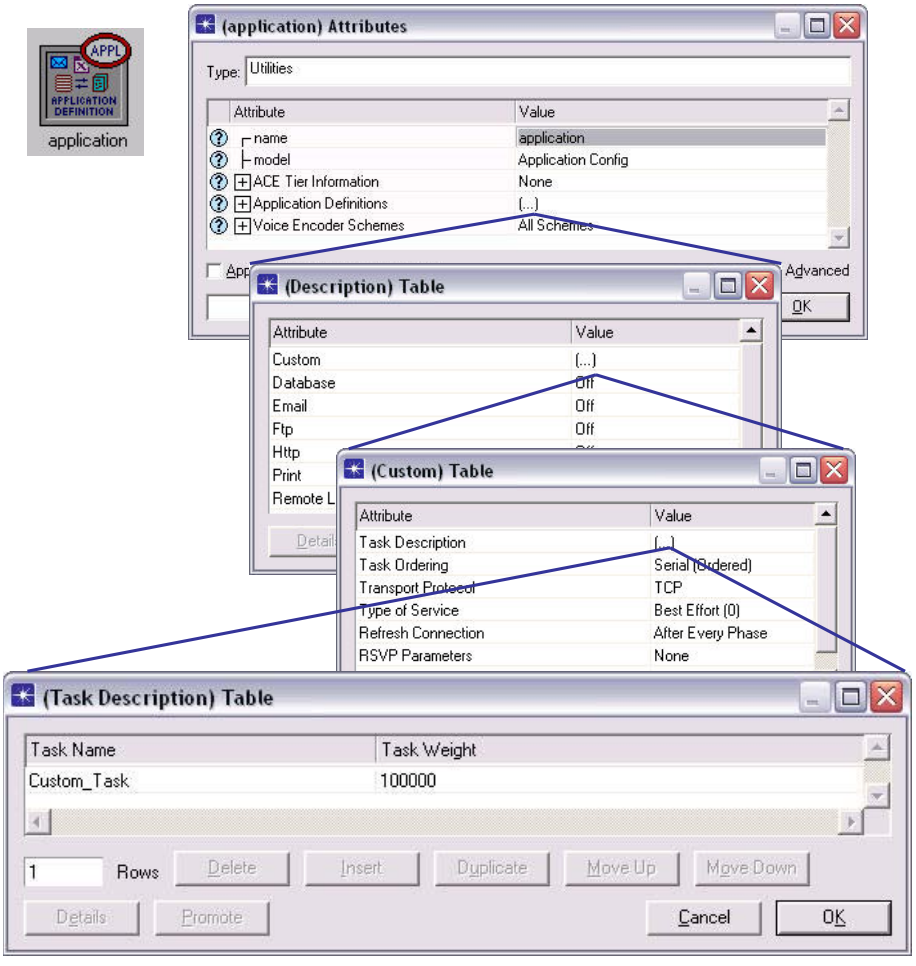


Figure 17. Sample configuration of application definition

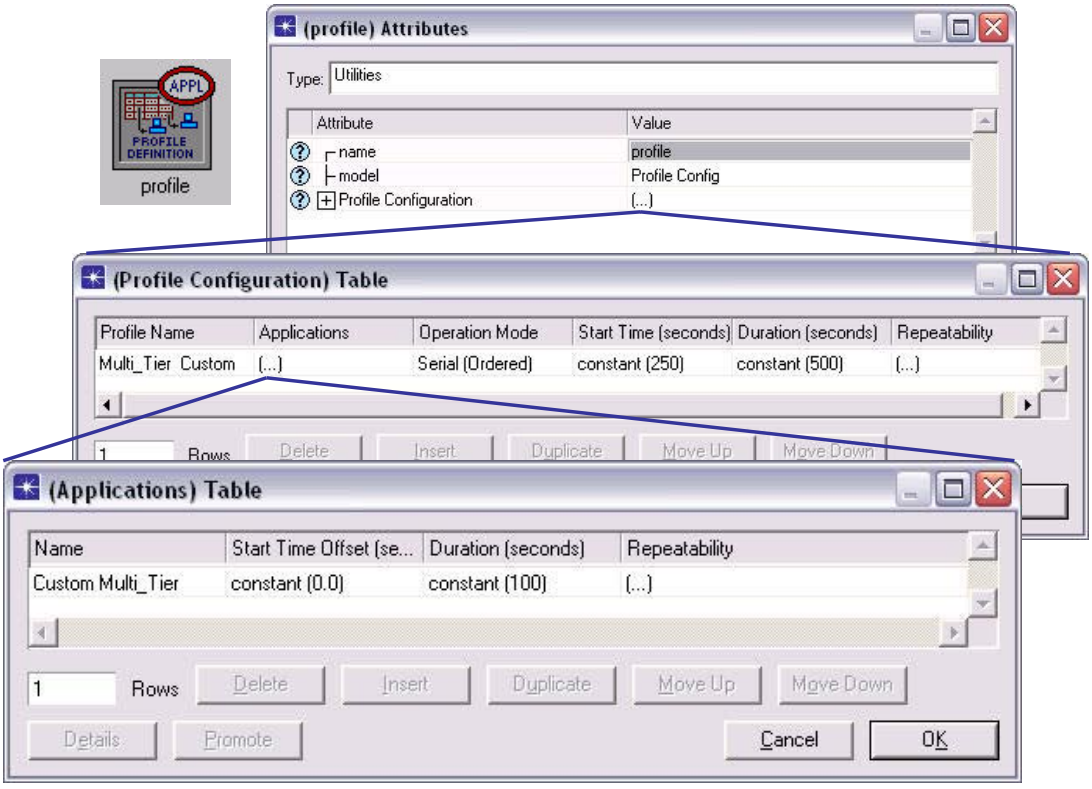


Figure 18. Sample configuration of profile definition

3. Collecting Statistics and Viewing Results

The eventual goal of simulation is to evaluate some aspects of a system's behavior or performance. Once the simulation models are built, the next logical step is to collect the required statistics and view the results in order to gain insight into the dynamic operation of the models. A statistic is a numerical variable representing a particular type of data related to the behavior of a node, link, or an entire system. Some statistics available in OPNET include:

- Inter-arrival times and packet sizes.
- Throughput, utilization, error rates and collisions.
- Application-specific statistics defined by a model developer.

After running the simulation and collecting the statistics, the results can be viewed in a graphical format in OPNET, as shown in Figure 19. Two different output files can be plotted – output vector files and output scalar files. Output vector files are usually generated by the simulation to store time-series data that are statistics which vary

as a function of simulation time. In this case, the plotted graph describes one statistic, and how it changes within a specified simulation period. The horizontal (x) axis represents the simulation time, while the vertical (y) axis represents the measured value, as illustrated by the graph in Figure 19. OPNET also supports collection of results over multiple simulations under different configurations or operating conditions of the system, and these results are accumulated in the output scalar files. These scalar statistics are then plotted again one another – plotting scalar Y against scalar X shows the possible values of scalar Y for individual values of scalar X.

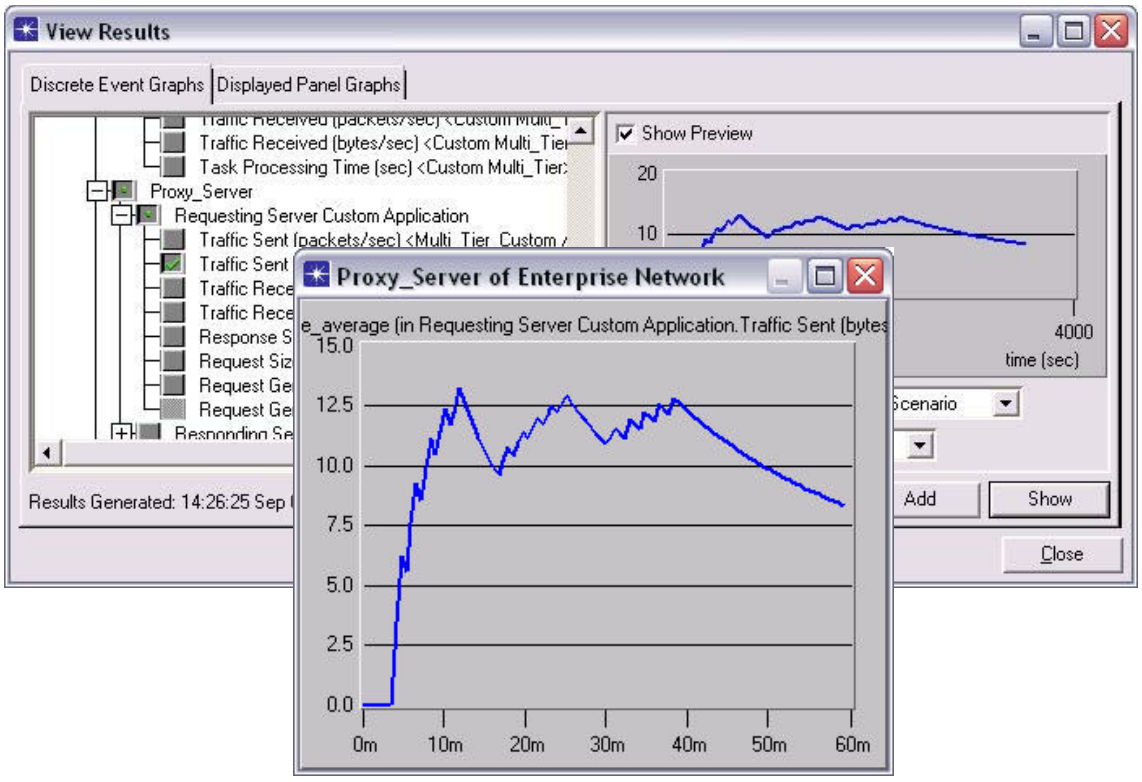


Figure 19. Statistic viewed in graphical format

B. OMNET++

OMNeT++ is a public-source, object-oriented modular discrete event simulator [Varga, 2003]. The name stands for Objective Modular Network Testbed in C++. The principal author of the simulator is Andras Varga, from the Technical University of Budapest, Department of Telecommunications (BME-HIT). The simulator can be used for a variety of applications, including:

- Traffic modeling of telecommunications networks.

- Protocol modeling.
- Modeling queuing networks.
- Modeling multiprocessors and other distributed hardware systems.
- Validating hardware architectures.
- Evaluating performance aspects of complex software systems.

1. Modeling Architecture

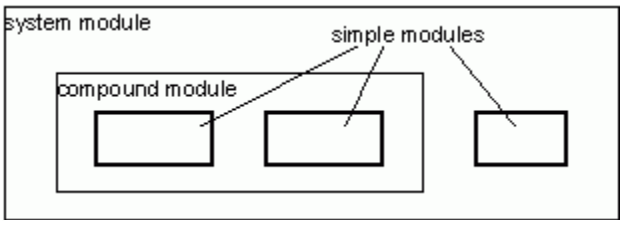
An OMNeT++ model is made up of hierarchically nested modules. The depth of the modules is not limited so that it can allow users to reflect the logical structure of actual system. Communication between modules is achieved through message passing. Messages can contain arbitrarily complex data structure. Modules send messages to their destinations either directly, or along a predefined path, through logical constructs known as gates and links. Modules are programmed in C++, which are assembled into higher-level modules, and then modeled using high-level language called NED (Network Description).

a. Hierarchical Modules

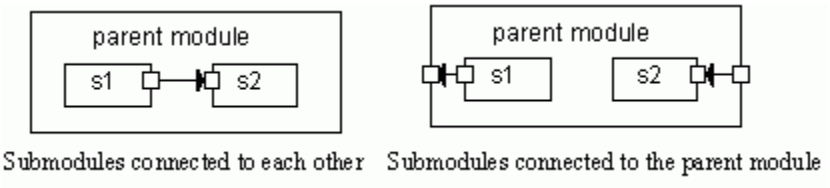
Figure 20(a) shows the modular nature of OMNeT++. The top level is the system module. The system module contains submodules, which can also contain other submodules. Modules that contain submodules are called compound modules, while the lowest level modules in the hierarchy are called simple modules. The simple module contains the algorithm in the model, programmed using C++ language.

b. Communication using Messages, Gates, and Links

Modules communicate by exchanging messages. Messages are communicated through input and output interfaces of the modules. These interfaces are known as gates. Links are used to connect the gates on modules. Each link is created within a single level of module hierarchy: within a compound module, one can connect the corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module (Figure 20(b)).



(a) Simple and compound modules



(b) Link connections

Figure 20. Hierarchical modules and link connections in OMNeT++ (From Ref [Varga, 2003])

Links can be assigned three parameters to facilitate the modeling of communications networks. These three parameters are propagation delay, bit error rate and data rate, all three being optional. Propagation delay is defined as the amount of time the message is delayed when traveling along a link. Bit error rate is defined as the probability that a bit is transmitted incorrectly, which is typical in a noisy communication channel. Data rate, which is specified in bits per second, is used in the calculation of transmission time of a packet.

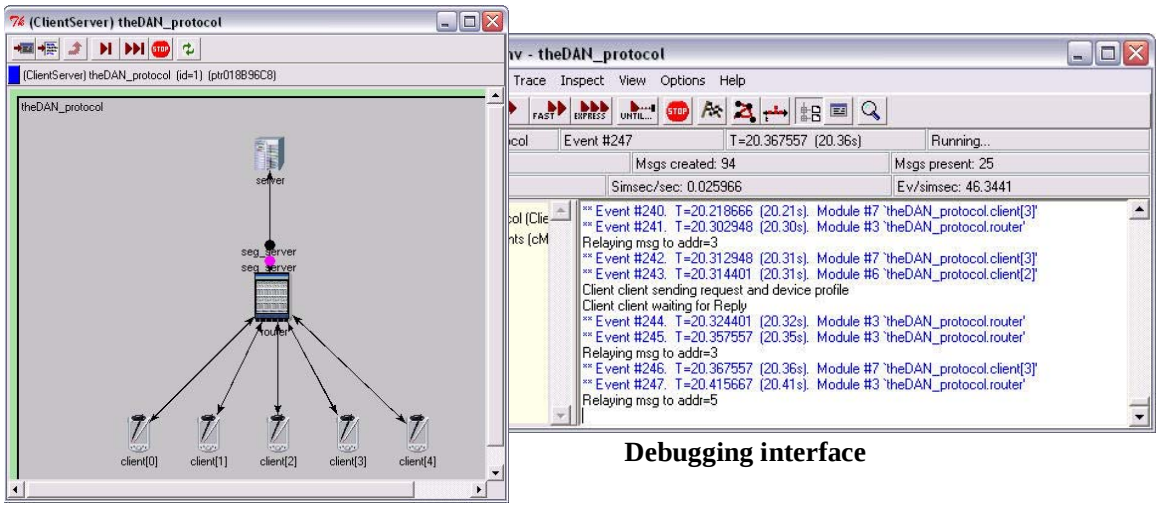
c. Topology Description Language

The topology of a model is specified using the NED language, which supports modular description of a network. This means that a network description consists of a number of component descriptions such as simple and compound modules, gates and links. The design of a network topology in NED language can be achieved graphically through GNED (Graphical NED Editor).

2. Running the Simulation

The simulation is run from a standalone executable program. When the program is executed, it reads in settings from a configuration file called omnetpp.ini. The configuration file defines how the simulation is run and the model parameter values.

OMNeT++ features various user interfaces for different simulation tasks, such as debugging, demonstration and batch execution. Such advanced user interfaces allow the user to visualize the inside of a model, to start/stop simulation execution, and possibly allow changing of variables/objects inside the model. These interfaces are illustrated in Figure 21.



Demonstration interface

Debugging interface

Figure 21. Demonstration and Debugging user interfaces in OMNeT++

3. Analyzing the Results

The output of the simulation is written into output files in the form of vector files, scalar files, or user's own defined file types. These vector and scalar files are similar to those described for OPNET above. OMNeT++ provides a Graphical User Interface (GUI) tool named Plove to view and plot the contents of output vector files (see Figure 22). The output files are text files in a format that can also be imported into external programs such as Excel for statistical analysis and visualization.

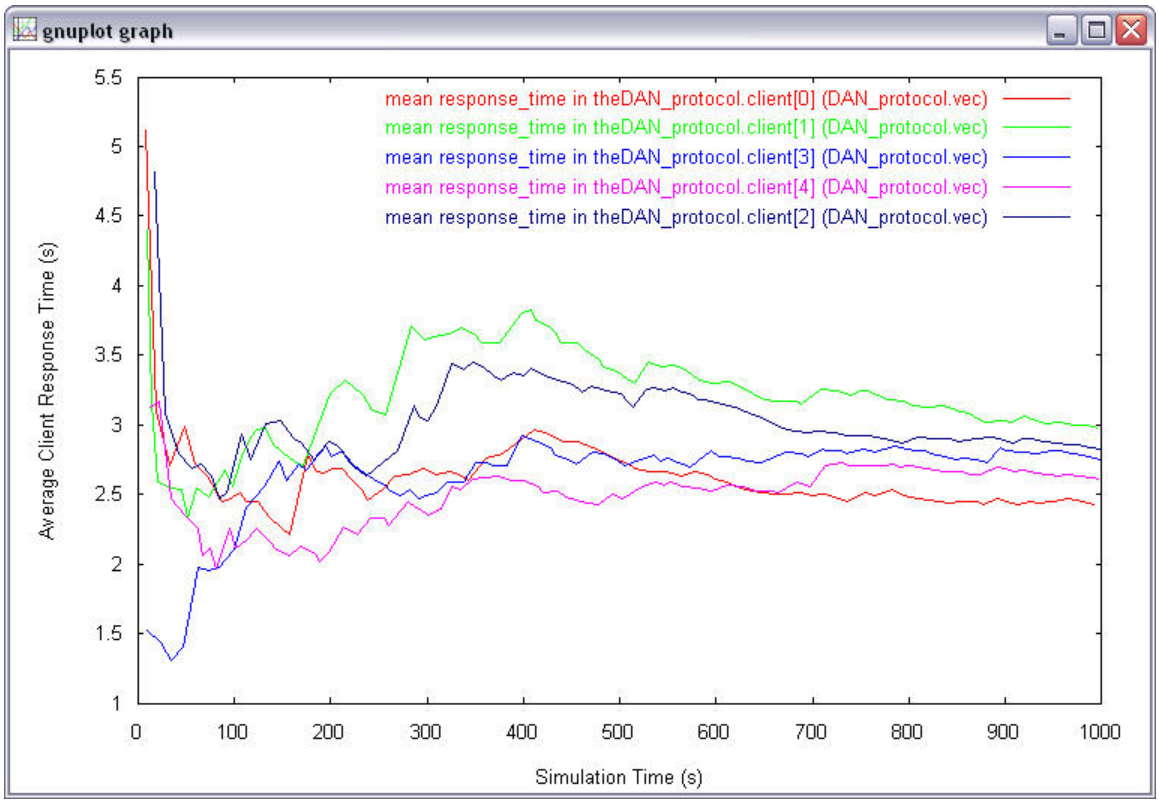


Figure 22. Plove view of output vector file

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PROPOSED DEVICE-AWARE NETWORK ARCHITECTURE

Several content repurposing and device profile delivery techniques are discussed in Chapter II. For repurposing, the techniques discussed in Chapter II provided content adaptation either at client, server or an intermediate proxy. Such techniques result in extra computational processing load at the already resource-limited client and at the server. They are also not optimizing the usage of network bandwidth as bandwidth-hungry content has to traverse across the network to be adapted at the client end, or at the intermediate proxy, especially when the content server is located a long distance away. These limitations will be illustrated through simulation modeling in Chapter V. In the case of device profile delivery, UAProf requires that the content server connects to a device profile repository to download the profiles as XML documents, which introduces additional undesirable network latency.

In this chapter, an architecture is proposed for DAN from an integrated systems approach, whereby the concept of a DAN processing unit (DPU) is introduced. The architecture consists of three functional components – encapsulating device profile information in the transmitting packet in binary-encoded format for small footprint and to avoid unnecessary network latency communicating with another device profile repository; adding content repurposing functionality into existing intermediate network entities, such as routers closest to content sources, which are called DAN processing unit (DPU), to optimize network bandwidth usage; capability-content compatibility policies to determine the need and extend of content repurposing to perform.

The next section describes the encapsulation of device profile within data packets. It is followed by Section B, which discusses the flexibility offered by DAN by incorporating content repurposing functionality into intermediate network entities. Section C provides an overview of capability-content compatibility policy engine. And finally, Section D discusses the advantages of the proposed architecture. Putting these functional components together, the proposed architecture provides an integrated systems approach to enable device-aware networking. The schematic of the proposed architecture is shown in Figure 23.

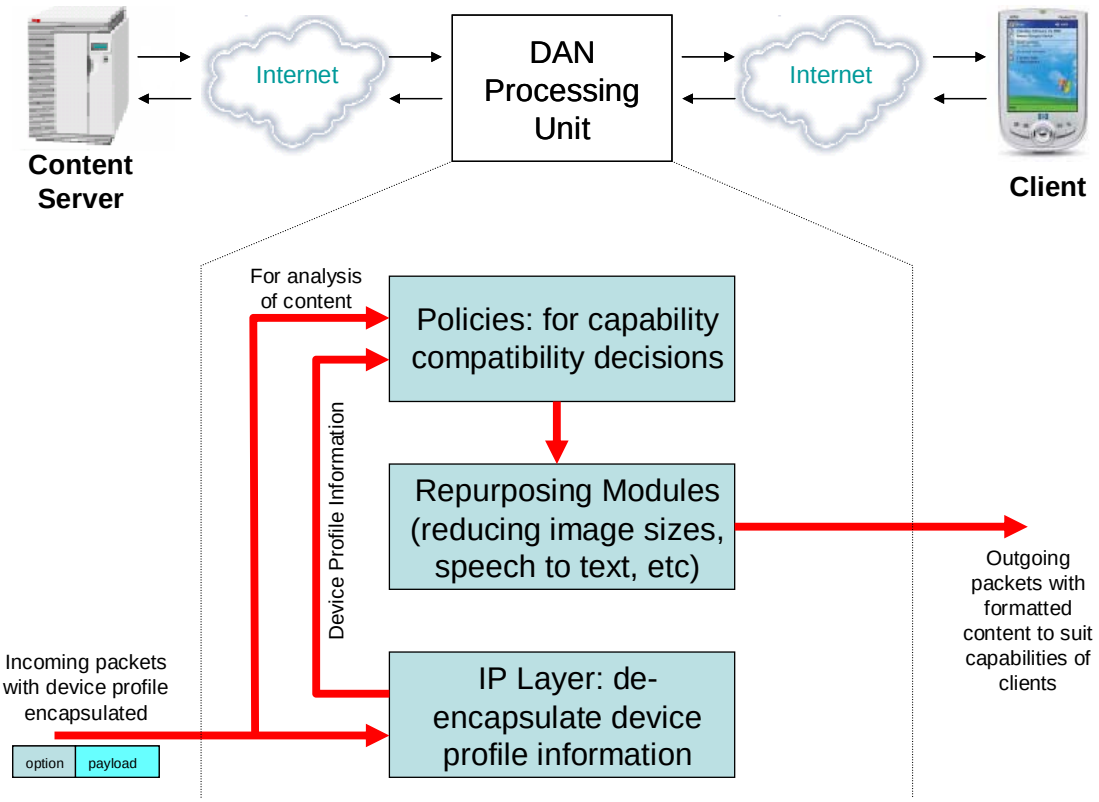


Figure 23. Schematic diagram of proposed architecture for DAN system

A. DEVICE CAPABILITY DISCOVERY

Several techniques are currently available to describe and communicate device capabilities, such as CC/PP [CC/PP, 2004] and UAProf [WAG, 2001] which are discussed in Chapter II. In particular, the device capability discovery mechanism supported by UAProf includes storage of descriptions as substantially large and comprehensive XML documents in networked databases, and retrieval of these documents over the network. These methods are not ideal. To avoid overhead of transmitting large documents and excessive information which may not be relevant to a device's ability to handle content, DAN architecture proposes encoding of device capability information in binary format, and encapsulation of the information within transmitting packets. In addition, only capability information crucial to determining content compatibility should be included in the description. IP Option field in IP header is considered as the encapsulation option for DAN architecture.

The structure of the IP Option format is shown in Figure 24 [Tcpipguide, 2004]. Of interest is the *Option Type* subfield, which is an 8-bit field divided into three "sub-subfields". *Option Class* sub-subfield is 2-bit long and specifies one of four potential values that indicate the general category into which the option belongs. Currently, only two of the values are defined: '0' for *Control* options, and '2' for *Debugging and Measurement* options. *Option Number* sub-subfield is 5-bit long and specifies the kind of option (32 different values) from each of the option classes. A few of the commonly employed are '0' used in military to indicate security classifications; '3' for loose source route; '7' for record route; '9' for strict source route; '4' for timestamp; '18' for traceroute; and so on.

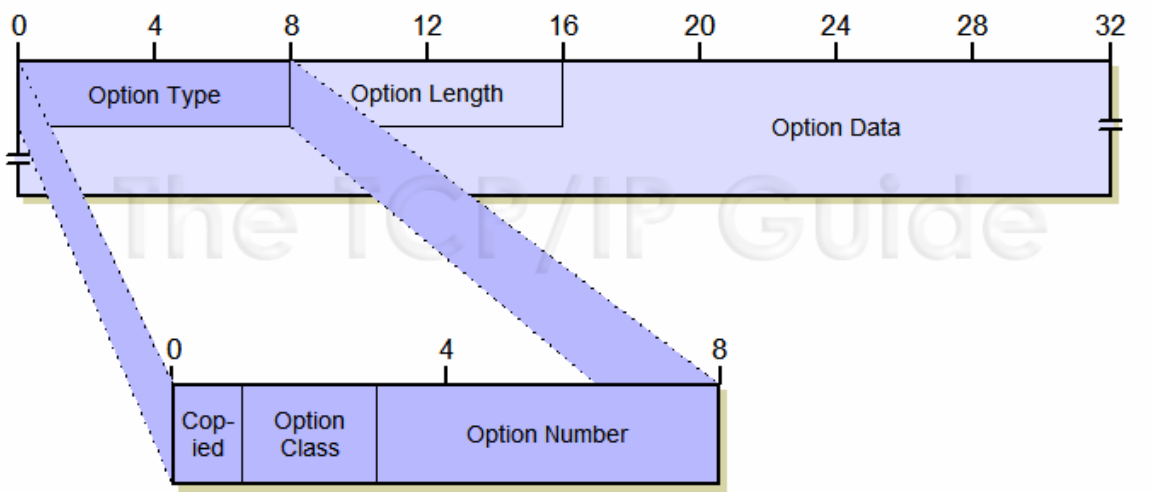


Figure 24. Structure of IP Option format (From Ref [Tcpipguide, 2004])

In the case of DAN, another *Option Class* and *Option Number* can be defined to indicate that the subsequent option data contains device capability information. The format of the option data can be a fix-sized field to represent the various types of capabilities, followed by blocks of variable-length fields to represent the attributes for the capability types. Common capability types may include CPU processing power, display screen size, sound output capability, type of multimedia encoders the device supports, and so on. Take for example; an 8-bit field is used to represent the various types of capabilities and the 2nd bit indicates the hardware display screen size. The bit is set and another variable-length field indicates the attribute which represents size of the device's

screen in units of pixels, composed of the screen width and the screen height dimension that can take values of “240x320”, “640x480”, etc. This example is depicted in Figure 25.

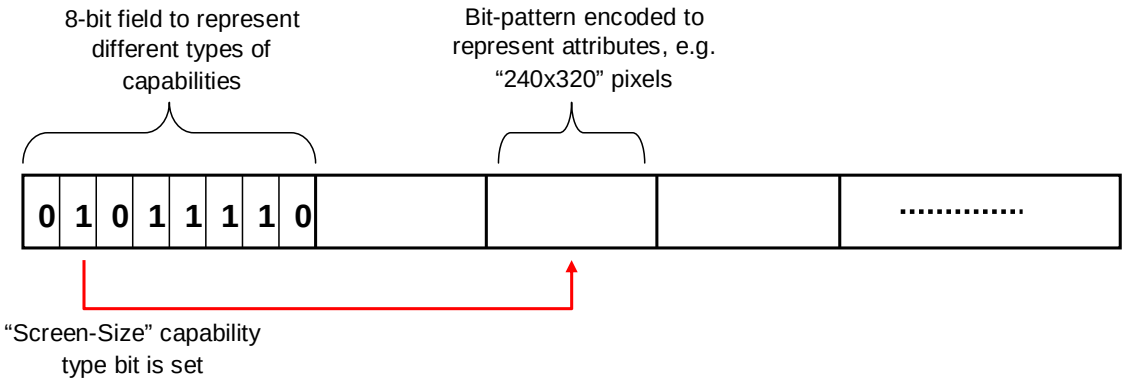


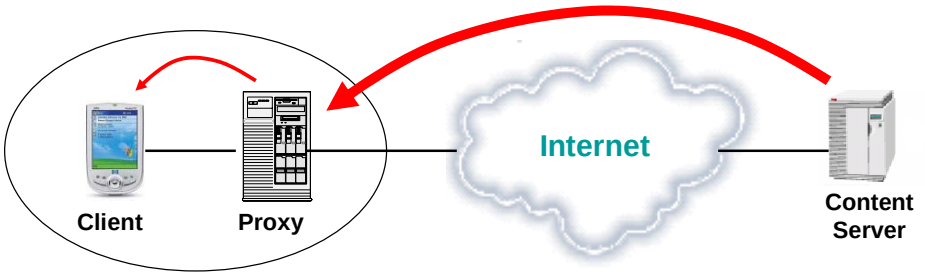
Figure 25. Example of DAN format to represent device capability information

In order to enable DAN, end host systems need to allow encapsulation of device capability information in the IP Option field in IP header. A DAN-enabled host can then easily transmit device capability information together with any request packet to the content server, without much impact on response time, as compared with the additional network latency incurred by UAProf. Chapter V documents the minimal impact on response time when IP Option field is used, through simulation modeling.

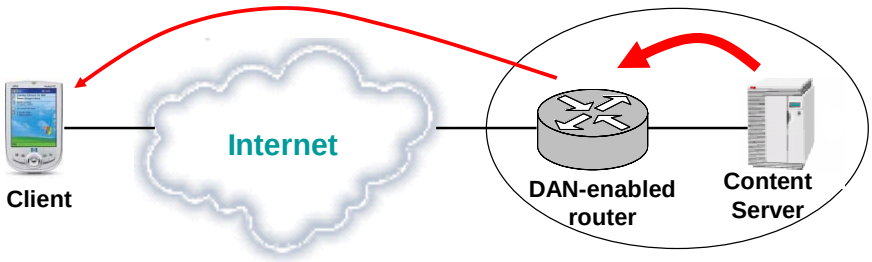
B. CONTENT REPURPOSING FUNCTIONALITY IN DAN PROCESSING UNIT (DPU)

The proposed DPU in the architecture can reside in either end hosts (clients and servers) or in existing nodes, e.g. routers, along the data path in the network. In this way, DAN offers a much more flexible architecture compared to current content repurposing frameworks presented in Chapter II. This flexibility is particular useful in mobile ad-hoc network environment, whereby both the clients and servers may be mobile devices with limited processing resources. These devices will not be able to perform the computationally intensive content repurposing operations. A more capable DPU along the data path can perform these operations so that only usable content is delivered to the client devices.

Preferably, the DPU should be located closest to the content sources, so that maximum network bandwidth conservation can be achieved. Canali et al proposed an enhanced proxy server prototype that integrates content adaptation functionality and caching of Web resources using Squid server [Canali et al, 2003]. Traditionally, proxy caching servers are located closest to the clients so that content can be cached to avoid re-fetching the content from the network if similar content is requested, thus conserving bandwidth. However, from DAN perspective, this approach is not optimal as unusable or huge content has to traverse across the network in its original size all the way from the content server to the proxy, only to have parts or all of the information discarded or reduced, utilizing network bandwidth unnecessarily. DAN proposes DPU functionalities to be incorporated into existing network entities such as routers closest to the content servers. Integrating with the existing caching service provided by the proxies at the client ends, network bandwidth conservation is maximal as size and frequency of information traversing majority of the network is reduced to minimal. Comparison between the two approaches is illustrated pictorially in Figure 26. Also, Chapter V will provide an insight numerically, through simulation modeling, on the improvement in network bandwidth utilization when DAN architecture is used compared to proxy approach.



(a) Proxy-approach: higher network utilization over the network



(b) DAN-approach: lower network utilization over the network

Figure 26. Comparison between proxy-based and DAN-based approach

With reference to Figure 23, DAN-enabled router performs three operations. Firstly, it will extract the device profile information encapsulated in the transmitted packets. Using this device profile information and the payload content, it will determine whether the content is compatible with the device's capabilities, making use of pre-defined policies. After determining capability-content compatibility, if the content is deemed compatible, it will pass through without additional processing. Otherwise, the content will be repurposed before forwarding the "usable" content to the client devices.

C. CAPABILITY-CONTENT COMPATIBILITY POLICY ENGINE

As illustrated in Figure 23, DAN architecture is modular in nature, with a separate policy engine for determination of compatibility between device capability and content. Besides using device capability information encapsulated within the transmitting packets as a factor for consideration, Han et al also proposed to use DPU-server bandwidth, DPU-client bandwidth and user preferences as factors to determine the need and the extend of content repurposing to perform [Han et al, 1998]. The modularity nature of DAN architecture offers the flexibility of feeding any number of parameters into the policy engine for decision-making.

There is continuing research on decision-making criteria for content repurposing, such as Hu's and Bugga's study on the functional categorization of web images [Hu and Bagga, 2004]. Identifying the functional categories is an important consideration for content repurposing. In this study, images are categorized as story, preview, host, commercial, icon logo and heading. Images that belong to categories such as commercial and icon logo can be safely removed without losing the meaning that content is bringing across, instead of going through image re-scaling process. Such decision-making criteria will continue to emerge as research on content repurposing progresses. Having a flexible and modular architecture that DAN proposed will facilitate new decision-making criteria to be easily included in the capability-content compatibility policy engine. Inevitably, analysis and processing delay will increase with new arguments added into the decision-making process. Chapter V will investigate the influence of such delay in overall DAN performance.

D. ADVANTAGES OF PROPOSED DAN ARCHITECTURE

Evident from the discussion above, the proposed architecture has several benefits over existing techniques presented in Chapter II. Firstly, it conserves network bandwidth and resources through having DAN processing unit located closest to the content sources so that the content can be repurposed at the earliest opportunity before traversing across the network, eventually reaching the client devices. Network bandwidth and resources are also conserved by encoding device profile information in binary format as compared to large XML documents, as practiced by UAProf; and encapsulating the information within the transmitting packets instead of incurring network latency and utilizing network resources by performing additional lookup process to retrieve the XML documents from networked repositories.

Secondly, DAN architecture is modular in nature and offers flexibility in deployment. The operations performed by DAN processing unit are divided into three modules – extraction of encapsulated device profile information; content repurposing operations; and the capability-content compatibility decision-making process through pre-defined policy engine. Modifications to the different modules can be easily incorporated if new and refined techniques are discovered as research in these areas continue to progress. In addition, DAN processing unit has the flexibility to reside in any existing entities along the data path in the network. This is particularly important in mobile ad-hoc network environment. In such environment, end hosts may be less capable to handle any content repurposing and more capable intermediate nodes can assist to perform the tasks. Data paths change frequently in a highly mobile environment, and the flexibility in the location of DPU can provide at least an intermediate node to perform the necessary content repurposing.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION MODELING AND PERFORMANCE EVALUATION

This chapter describes the design, development and implementation of simulation models used to study some of the performance limitations of existing content repurposing frameworks and device capability discovery techniques, and to compare them with the proposed DAN architecture. The various parameters used for the simulation and analysis of the simulation results are also presented.

A. SIMULATION MODEL A: CLIENT-BASED AND SERVER-BASED REPURPOSING

This model is a client-server architecture with the nodes communicating over the Internet, as shown in Figure 27 below. The model is developed using OPNET Modeler simulation software.

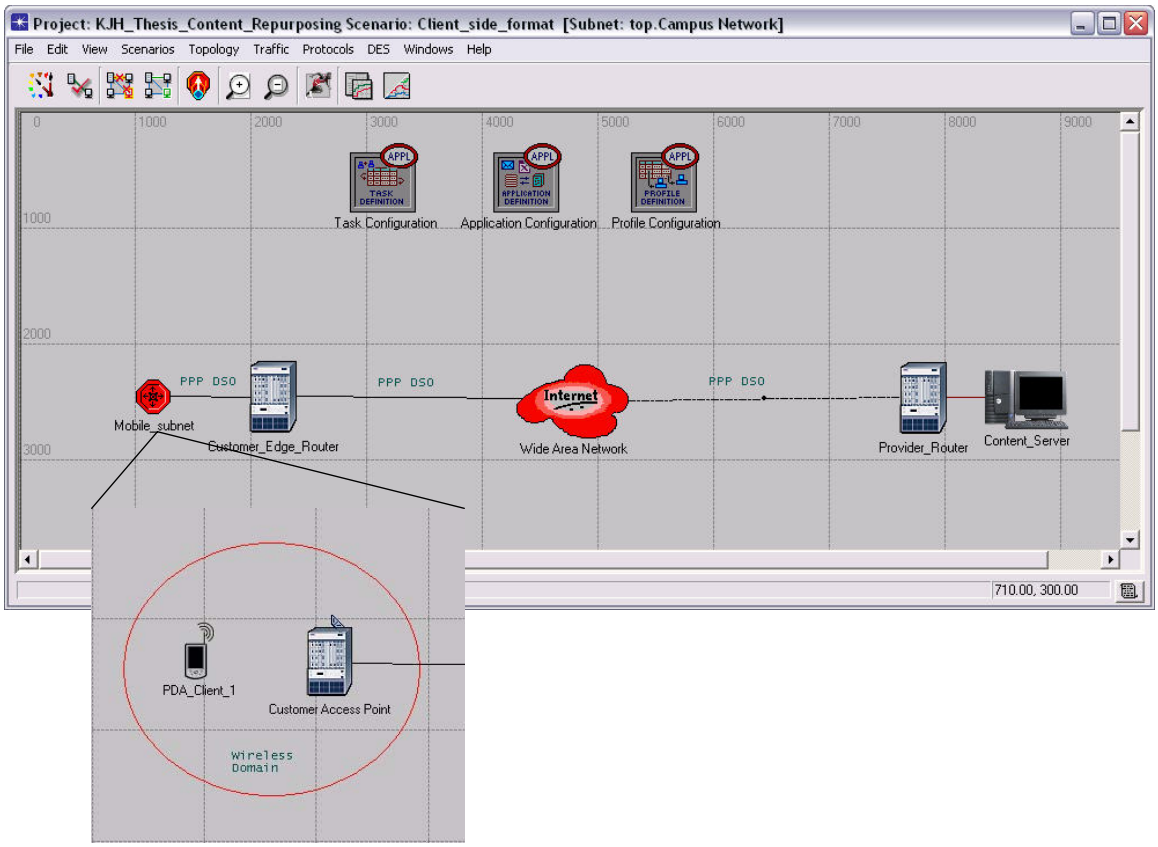


Figure 27. Simulation model for client-server communication

The server node represents a content server serving content requested by clients. The point of attachment to the Internet is a DS0 – 64 kbps connection via a router. The client node represents either a normal PC or a PDA with limited display and processing capabilities, with 802.11b wireless connection at 1 Mbps data-rate to an access point. The client network is also connected to the Internet through a DS0 – 64 kbps connection via a router. The client node will periodically request for content from the content server.

1. Scenario Description

Three different scenarios are built using the model. The detailed descriptions of the scenarios are presented in Table 2.

Scenario	Description
Scenario 1: Baseline configuration	In this scenario, the client node is a normal PC with capabilities to handle the requested content. The content does not need to undergo any form of repurposing. The communications between the PC and content server follows a normal request-reply transaction flow using TCP (Transmission Control Protocol). Normal CPU processing delay is incurred at both the PC and content server. The values for the parameters used will be given in the next sub-section.
Scenario 2: Repurposing performed at client side	In this scenario, the client node is a resource-limited PDA. The content must be repurposed in order to be "usable" for the PDA to handle, and the system employs a client-based repurposing approach. The communications between the PDA and content server follows a transaction flow and processing delay depicted in Figure 28. This communication is configured as custom application using task specification in OPNET, with TCP as the transport protocol. The values of the parameters used are given in the next sub-section.

Scenario	Description
Scenario 3: Repurposing performed at server side	In this scenario, the client node is also a resource-limited PDA. The content must be repurposed in order to be "usable" for the PDA to handle, and the system employs a server-based repurposing approach. The communications between the PDA and content server follows a transaction flow and processing delay depicted in Figure 29. This communication is also configured as custom application using task specification in OPNET, with TCP as the transport protocol. The values of the parameters used will be given in the next sub-section.

Table 2. Descriptions of scenarios using client-server communication model

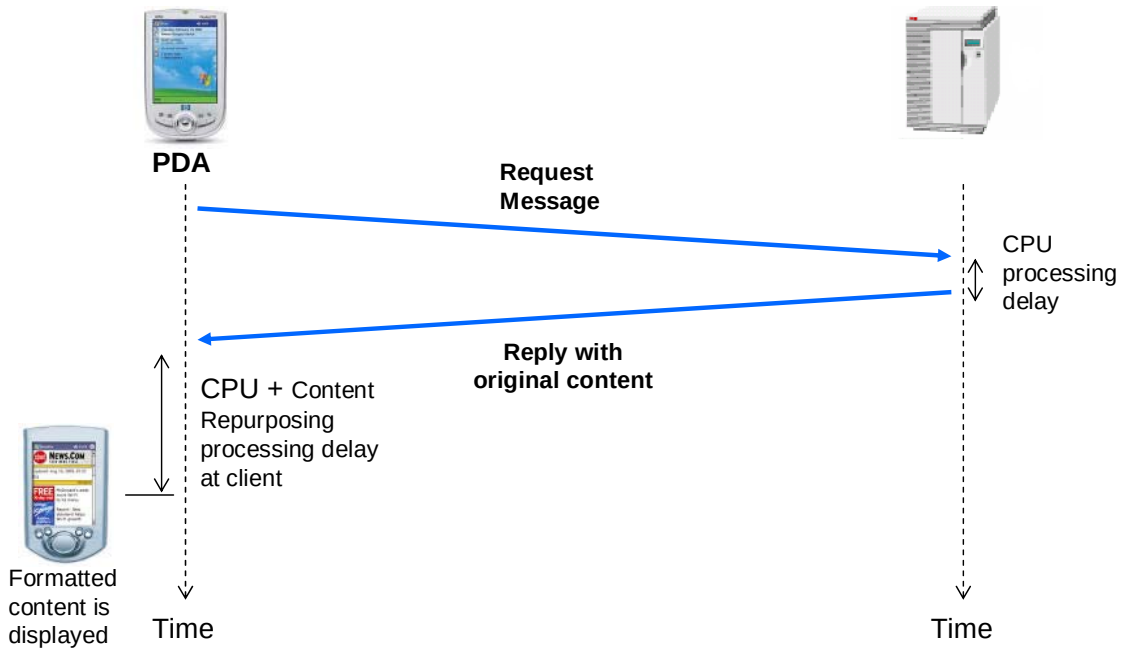


Figure 28. Transaction flow for Scenario 2: client-based repurposing

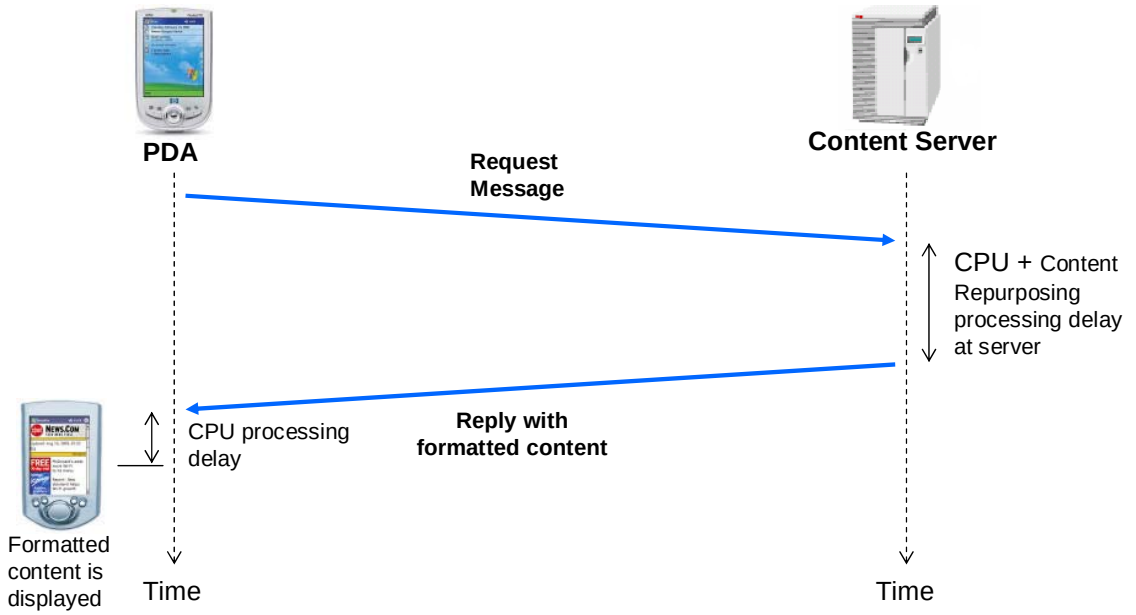


Figure 29. Transaction flow for Scenario 3: server-based repurposing

2. Task Definition and Parameters Used

The communications between the client node and server node for the scenarios are configured as custom applications. The entire process of requesting for content by the client node is described as a task. Each scenario has its own specified task as described in Table 2 above. Phases of message passing and processing delay for the various request and reply transactions are defined in Task Definition object in OPNET. Their definitions and the parameters used are provided in Tables 3, 4, and 5, respectively for the three different scenarios. Other delays such as propagation and transmission latency are inherently computed and taken into account by OPNET when running the simulation.

Scenario 1			
Phase	**Source**	**Destination**	**Traffic Description**
Transaction #1	PC	Content Server	Request message size is a uniform distribution over the range (1024, 2048) in bytes.
Transaction #2	Content Server	PC	CPU processing delay incurred at server node to generate reply is a uniform

Scenario 1			
Phase	**Source**	**Destination**	**Traffic Description**
			distribution over the range (0.05, 0.1) in seconds. Reply message size is a uniform distribution over the range (200000, 400000) in bytes.
Transaction #3	PC	Not Applicable	Internal processing delay incurred at PC to display content – uniform distribution over the range (0.05, 0.1) in seconds.

Table 3. Definition of phases in Scenario 1

Scenario 2			
Phase	**Source**	**Destination**	**Traffic Description**
Transaction #1	PDA	Content Server	Request message size is a uniform distribution over the range (1024, 2048) in bytes.
Transaction #2	Content Server	PDA	CPU processing delay incurred at server node to generate reply is a uniform distribution over the range (0.05, 0.1) in seconds. Reply message size is a uniform distribution over the range (200000, 400000) in bytes.
Transaction #3	PDA	Not Applicable	Delay incurred is a combination of CPU processing and extra processing for content repurposing in order to display formatted content – uniform distribution over the range (0.1, 0.3) in seconds.

Table 4. Definition of phases in Scenario 2

Scenario 3			
Phase	**Source**	**Destination**	**Traffic Description**
Transaction #1	PDA	Content Server	Request message size is a uniform distribution over the range (1024, 2048) in bytes.
Transaction #2	Content Server	Not Applicable	Delay incurred at server node is a combination of CPU processing and extra processing for content repurposing so that only formatted content is delivered to the client – uniform distribution over the range (0.1, 0.3) in seconds.
Transaction #3	Content Server	PDA	Formatted reply message size is a uniform distribution over the range (150000, 200000) in bytes.
Transaction #4	PDA	Not Applicable	Internal processing delay incurred at PDA to display content – uniform distribution over the range (0.05, 0.1) in seconds.

Table 5. Definition of phases in Scenario 3

In a study conducted by IBM T.J. Watson Research Center, image transcoding delay was found to be dependent on input size, image dimensions, image content, transcoding parameters, and compression and de-compression algorithms [Han et al, 1998]. Delay due only to image transcoding process was also predicted to be in the range of 50 – 200 ms for JPEG-to-JPEG conversion, and exhibited a linear correlation with the area of the input image (number of pixels). The results from the study are reproduced in Figure 30, for the cases of quality factor = 5, correlation coefficient = 0.98, and quality

factor = 50, correlation coefficient = 0.98. These results are assumed for processing delay parameter values due to content repurposing in the simulations for this thesis report. The selection of values for pre-formatted and formatted message size parameters will be elaborated in Section B.

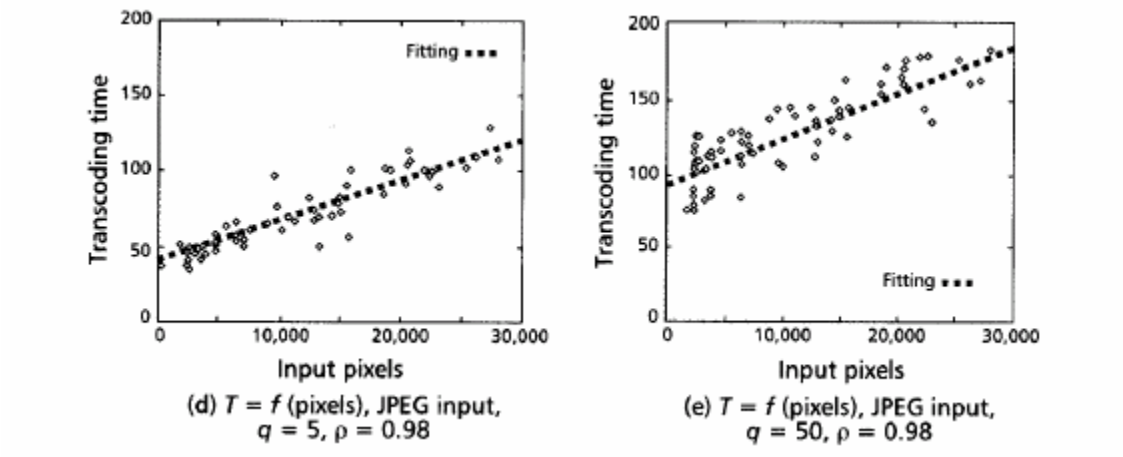


Figure 30. Prediction of image transcoding time (in ms) for a transcoded image (From Ref [Han et al, 1998])

A sample configuration, in OPNET of the phases described in Table 5 for Scenario 3 is shown in Figure 31. Also shown is the detailed configuration of the traffic parameters for one of the phases.

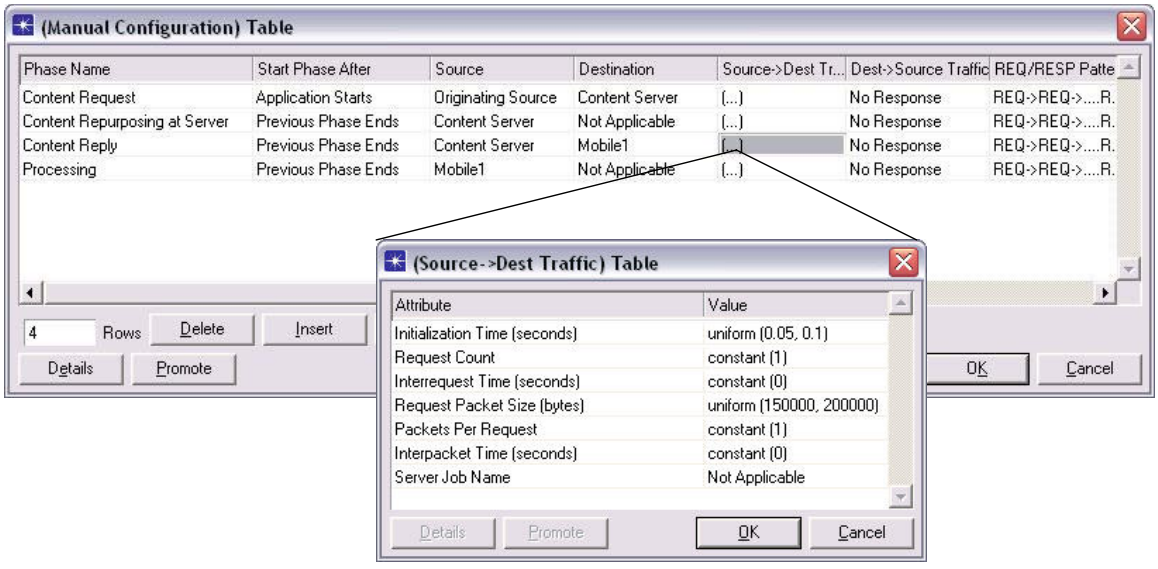


Figure 31. Sample configuration of phases and traffic description in OPNET for Scenario 3

3. Profile Definition and Running the Simulation

The profile used by the client node in the scenarios is configured using the Profile Definition object. The client node is configured to initiate a request to the content server periodically with inter-request time following an exponential distribution with mean time of 180 s.

The simulations are set to run over a simulation period of 120 min. Enhanced Interior Gateway Routing Protocol (EIGRP) is used as the routing protocol running in the network, so that packets in the network can be routed correctly to their destinations.

B. COMPARING SIMULATION RESULTS FOR SIMULATION MODEL A

Several significant statistics are collected from the simulation. The graphical results of the simulation model for CPU utilization of the content server, CPU utilization of the client, and network utilization over the Internet between the two nodes are shown in Figure 32, 33, and 34, respectively. Time average values are plotted for the statistics. Time average value is represented by the expressions (*sum+=x; x=sum/++n*), where *x* is the instantaneous value, and *n* is the number of statistics collected over the simulation period.

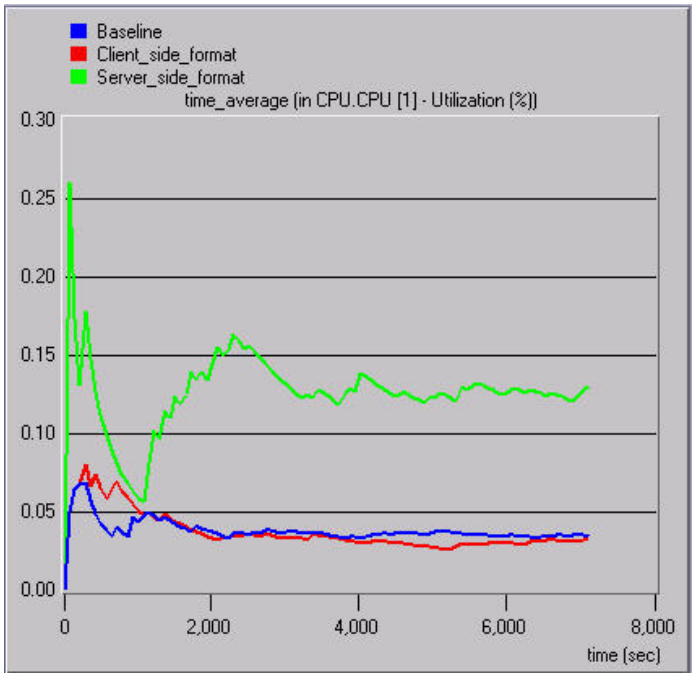


Figure 32. Comparing CPU utilization of content server for 3 different scenarios

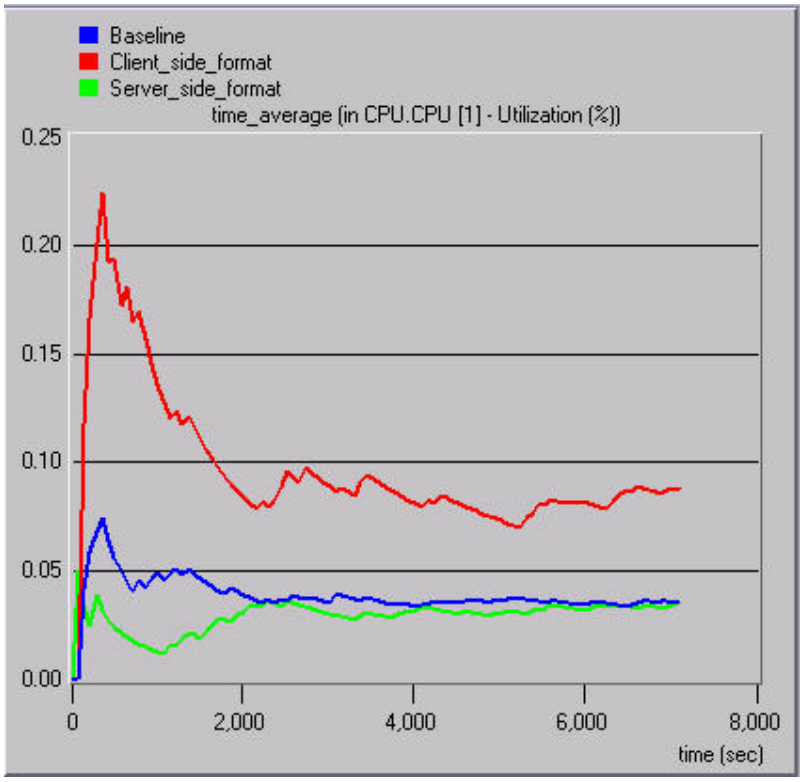


Figure 33. Comparing CPU utilization of client node for 3 different scenarios

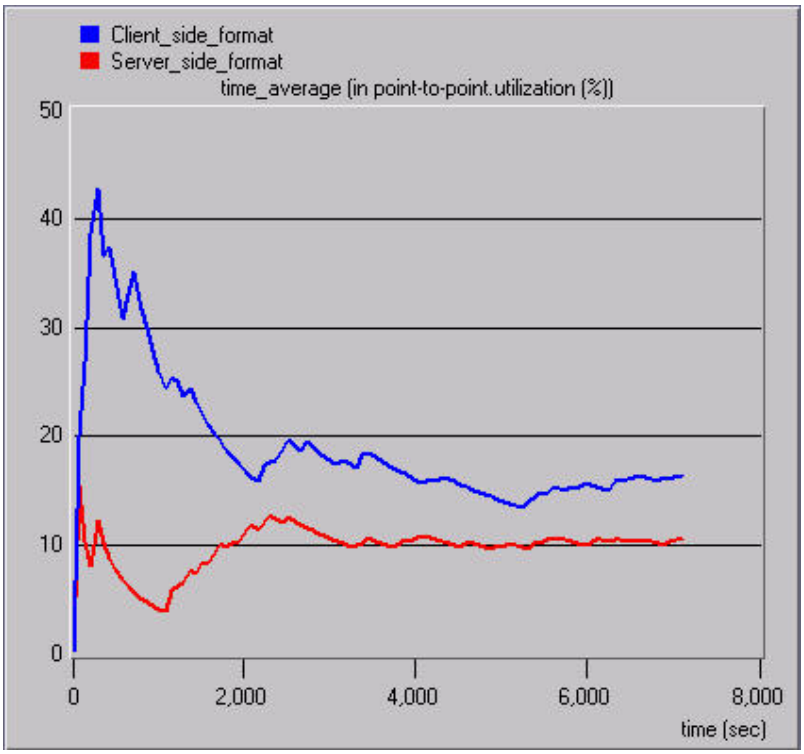


Figure 34. Comparing network utilization for client-based and server-based repurposing approaches

The CPU utilization diagram for content server (Figure 32) shows that when content repurposing is performed at the server, the extra computational load increases CPU utilization by a factor of 3.25, from an average value of 0.04% to 0.13%. This significant increase in CPU loading will affect the server's ability to serve more clients. Similarly, if content repurposing is performed at the client, the additional computational load increases the CPU utilization of client by a factor of 2, from an average value of 0.04% to 0.08% (Figure 33). Furthermore, client-based repurposing approach is not an efficient technique from the network bandwidth perspective as it consumes more bandwidth as compared to the server-based repurposing approach (Figure 34). Network utilization for client-based approach is 17%, while that for server-based approach is 10%. The simulation results demonstrated and reinforced some of the significant limitations of the two approaches to content repurposing.

C. SIMULATION MODEL B: PROXY-BASED REPURPOSING AND DAN ARCHITECTURE

This model is designed based closely on the setup of AvantGo service [AvantGo, 2004], which is an excellent example of proxy-based repurposing system, described in Chapter II. Six randomly selected channels (websites), as listed in Table 6, are used in building this model. These channels fall into categories such as news, entertainment, sports, technology and travel. They cover a broad range of interests, which form a typical profile of a user of AvantGo service.

Internet traceroute to the various websites was conducted, and from the results, the locations of the web servers (from network connectivity perspective) are gathered and recorded in Table 6. A sample traceroute result to AccuWeather website is shown in Figure 35, indicating that the network path ended in a router located in San Francisco, CA. Based on these locations, the simulation model is developed as shown in Figure 36, using OPNET Modeler simulation software.

Central to AvantGo service is the AvantGo Sync server, which is the proxy performing all content repurposing operations before serving out the formatted content to users. It is situated in San Jose, CA (network connectivity sense). This proxy server is also included in the simulation model.

Channel Description	URL of Website	Location
New York Post	http://www.nypost.com	New York City, NY
Computer World	http://www.computerworld.com	Boston, MA
TV Guide	http://www.tvguide.com	Denver, CO
AccuWeather	http://www.accuweather.com	San Francisco, CA
Sporting News	http://www.sportingnews.com	Washington, WA
Hollywood Movies	http://www.hollywood.com	Miami, FL

Table 6. List of AvantGo channels selected for simulation model

Traceroute to accuweather.com from mcc-engr-01.inet.qwest.net

```
traceroute: Warning: ckecksums disabled
traceroute to accuweather.com (207.242.93.24), 30 hops max, 40 byte packets
 1  66.77.78.66 (66.77.78.66)  0.830 ms  0.619 ms  0.587 ms
 2  cntr-02.mcc.qwest.net (63.150.176.9)  0.628 ms  0.525 ms  0.516 ms
 3  205.171.214.5 (205.171.214.5)  4.166 ms  4.239 ms  4.139 ms
 4  205.171.14.82 (205.171.14.82)  4.641 ms  4.699 ms  4.637 ms
 5  205.171.214.138 (205.171.214.138)  4.842 ms  4.799 ms  4.752 ms
 6  qwest-gw.sffca.ip.att.net (192.205.32.81)  5.890 ms  5.847 ms  5.849 ms
 7  tbr1-p012201.sffca.ip.att.net (12.123.13.189)  6.523 ms  6.726 ms  6.804 ms
 8  tbr1-cl1.cgcil.ip.att.net (12.122.10.5)  50.594 ms  50.468 ms  50.297 ms
 9  ar1-p310.clboh.ip.att.net (12.123.210.1)  59.313 ms  59.198 ms  59.128 ms
10  12.126.242.186 (12.126.242.186)  75.935 ms  72.934 ms  72.392 ms
11  * * *
12  www.accuweather.com (207.242.93.24)  108.933 ms  73.160 ms  73.142 ms
```

Figure 35. Sample traceroute result to AccuWeather website

In the simulation model, each channel subnet is made up of the website server connected to the Internet via a router through a DS0 - 64 kbps connection, as shown in the blown-up diagram (Figure 36) for “tvguide.com” channel. In reality, the content provider will subscribe to a larger bandwidth connection. A DS0 connection is chosen for the simulation so that the impact on network bandwidth usage is visually more significant and observable. The relative results for proxy-based approach and DAN approach are similar even when higher bandwidths are chosen. The client subnet is made up of 5

mobile PDAs with 1 Mbps 802.11b wireless connections to an access point (shown in blown-up diagram for client subnet in Figure 36). The client subnet is connected to the Internet through a DS1 – 1.544 Mbps connection, located at Monterey, CA. Likewise, AvantGo Sync server is also connected to the Internet through a DS1 – 1.544 Mbps connection. The various subnets are positioned approximately at their respective locations on an USA map so that propagation delay can be estimated and taken into account based on distance.

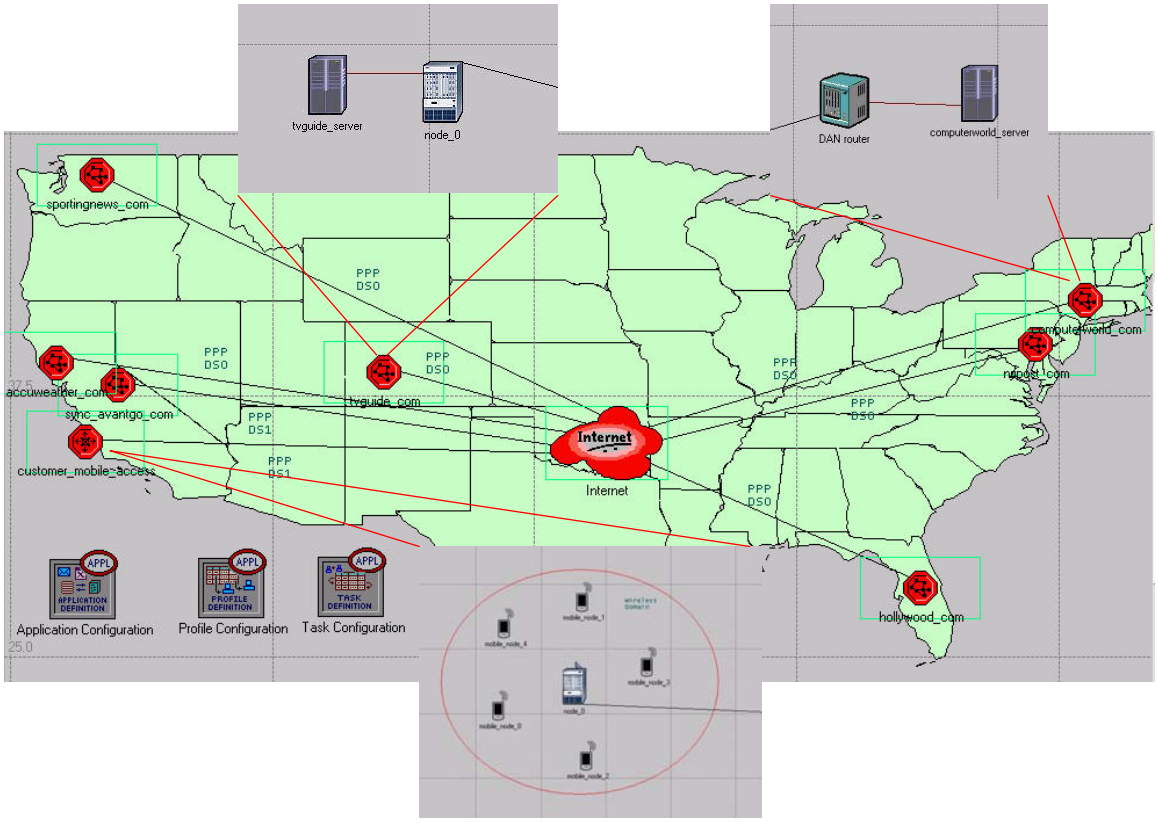


Figure 36. Simulation model used for proxy-based repurposing approach and DAN architecture

1. Scenario Description and Parameters Used

Two scenarios are built using the model. The first scenario models the AvantGo service – a proxy-based repurposing approach, and the second scenario models DAN architecture. Descriptions of the scenarios are provided in Table 7.

Scenario	Description
Scenario 1: Proxy-based repurposing	In this scenario, the PDAs will request for content from the various channel websites through the AvantGo Sync proxy server. The proxy server will fetch the necessary content in their original sizes from the respective websites, format the content, and subsequently sent the formatted content to the requesting client. The communications between the client, proxy server and website server follow a transaction flow and processing delay depicted in Figure 37. The communication is configured as a custom application in OPNET.
Scenario 2: DAN architecture	In this scenario, the router connecting the website server in each channel subnet is DAN-enabled, with the capability to perform content repurposing, as shown in the blown-up diagram in Figure 36 for "computerworld.com" channel. A client will request content directly from the website server. As the content traverse the network from the website server to the client, it will be intercepted by the DAN-enabled router, formatted before forwarding to the client. The communications between the client, website server and DAN-enabled router follow a transaction flow and processing delay depicted in Figure 38. The communication is also configured as a custom application in OPNET.

Table 7. Descriptions of scenarios for simulation model B

To get an estimate of the size of the original HTTP message returned from the respective websites, packet captures are conducted using Ethereal packet analyzer software. Only the message size of the homepage of the respective websites are captured and tabulated in Table 8. Figure 39 shows an example of the packet analysis for New York Post Homepage. Also, from AvantGo website (http://my.avantgo.com), information on the estimated and maximum size of data a user is expected to download when

requesting content from a specific channel, can be obtained, and they are tabulated in Table 8. This information is used as the assumed parameter values for original content size and formatted content size, respectively when accessing the different websites, in the simulation.

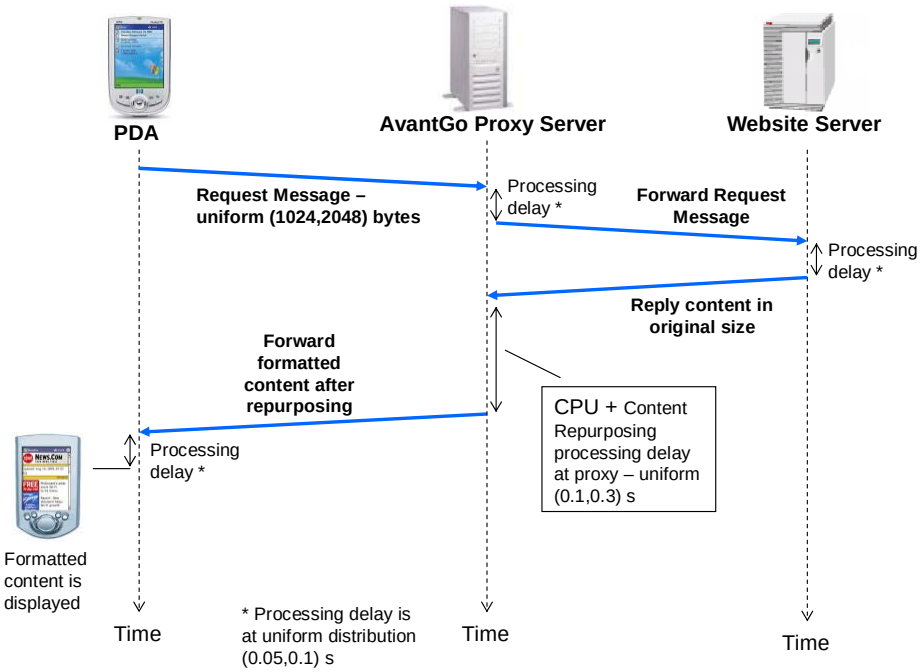


Figure 37. Transaction flow for Scenario 1: proxy-based repurposing

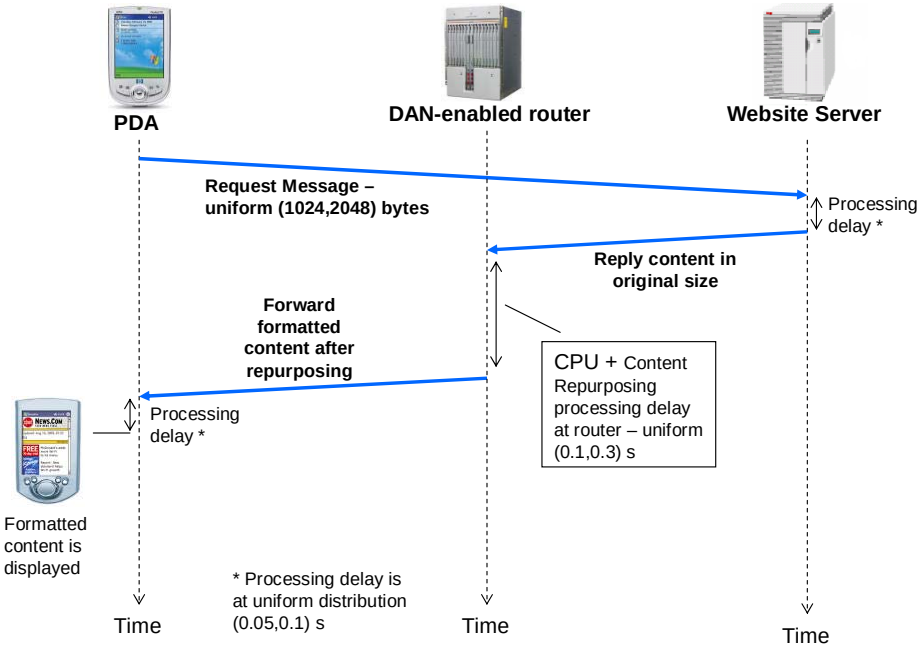


Figure 38. Transaction flow for Scenario 2: DAN architecture

Channel	Estimated size provided by AvantGo	Maximum size provided by AvantGo	Homepage size (captured through Ethereal)	Formatted size range (used in simulation)	Original size range (used in simulation)
Nypost	150 KB	200 KB	336760 B	150-200 KB	300-350 KB
Computerworld	25 KB	75 KB	353246 B	25-75 KB	350-400 KB
Tvguide	20 KB	100 KB	396080 B	20-100 KB	350-450 KB
Accuweather	150 KB	200 KB	272693 B	150-200 KB	250-300 KB
Sportingnews	100 KB	1500 KB	439481 B	100-350 KB	400-450 KB
Hollywood	25 KB	100 KB	402128 B	25-100 KB	400-450 KB

Table 8. Message sizes of different channels used in simulation (Homepage size is captured through Ethereal on 6 October 2004)

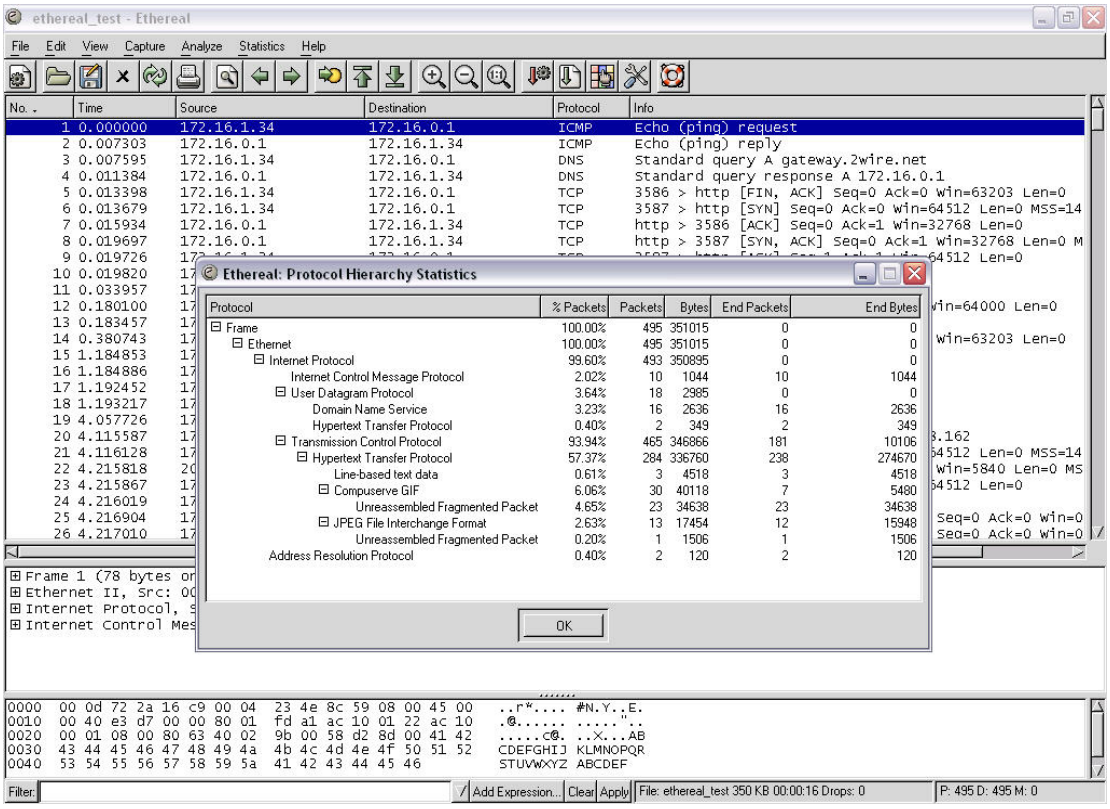


Figure 39. Packet analysis of New York Post homepage in Ethereal

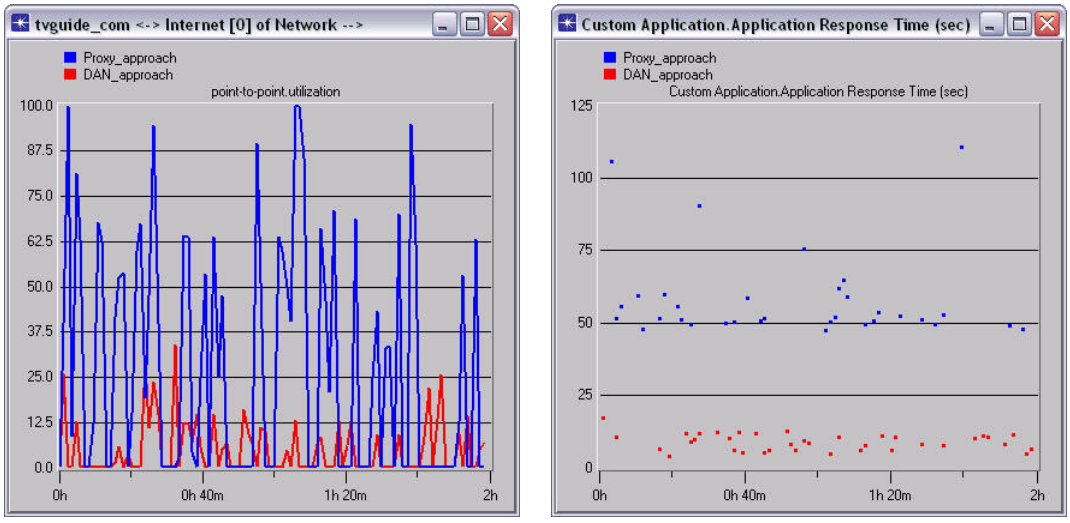
2. Profile Definition and Running the Simulation

Two profiles are created for the clients using Profile Definition object. One profile (Profile A) has the clients accessing to Accuweather, Computerworld and Hollywood website servers periodically at an interval following an exponential distribution with mean time of 300 s. The other profile (Profile B) has the clients accessing to Sportingnews, Tvguide and Nypost website servers periodically at an interval following an exponential distribution with mean time of 300s. For the simulation, three clients are assumed to be assigned Profile A, and two clients are assigned Profile B.

The simulations are set to run over a simulation period of 120 min. Enhanced Interior Gateway Routing Protocol (EIGRP) is used as the routing protocol running in the network, so that packets in the network can be routed correctly to their destinations.

D. COMPARING SIMULATION RESULTS FOR SIMULATION MODEL B

Network bandwidth usage and application response time are two important statistics that measure and compare the performance of proxy-based repurposing approach and the proposed DAN architecture approach. These statistics are collected from the simulations. Figure 40 shows an example of the graphical simulation results for (a) the network utilization on the connection between Tvguide server and the Internet, and (b) the response time experienced by the clients accessing Tvguide server. Response time is the total round-trip time taken for the clients to complete the transaction flow depicted in Figure 37 or Figure 38, depending on the approach used.



(a) Network Utilization (%) (b) Response Time (s)

Figure 40. Sample simulation results of network utilization and response time

The average network utilizations and response times for the six different channel (website) servers using different repurposing approaches (proxy-based and DAN) are compared graphically in Figures 41 and 42, respectively.

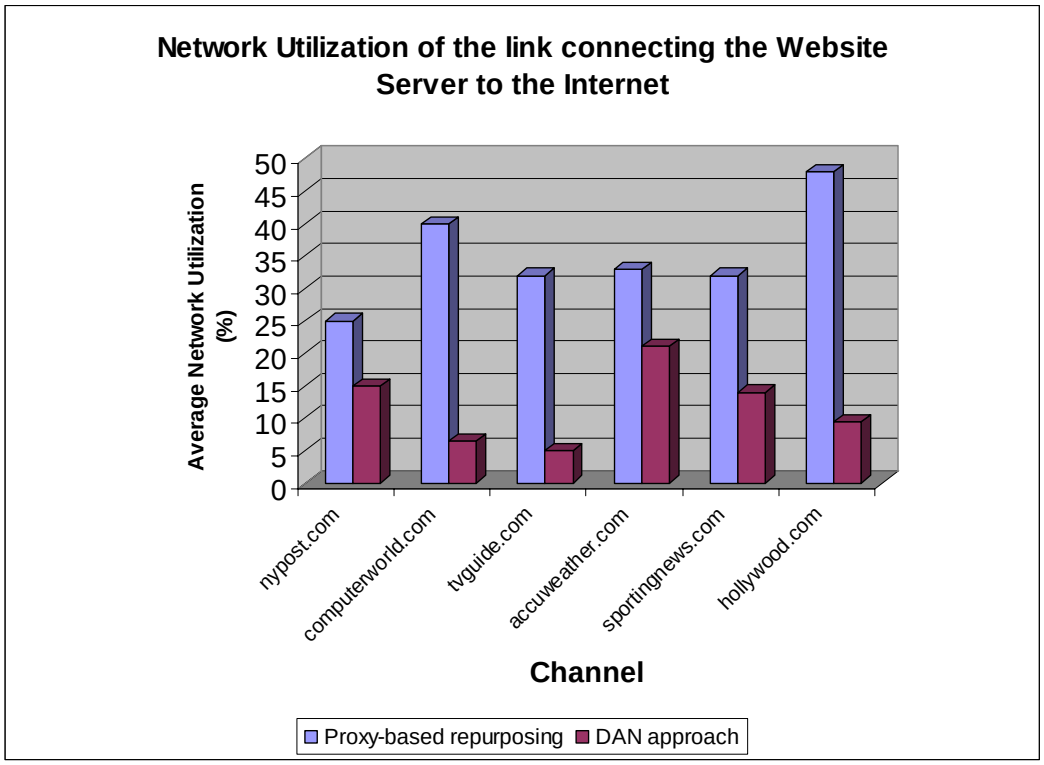


Figure 41. Comparing network utilization between proxy-based repurposing and DAN approach

As observed in Figure 41, network utilizations of the various links are reduced substantially when DAN approach is utilized, in comparison with proxy-based repurposing approach. The maximum reduction is experienced in the case of Hollywood channel (38.5%) and the minimum reduction for Nypost channel is 10%. This is the result of DAN approach incorporating repurposing functionalities in routers closest to content sources, so that only formatted content will traverse the network.

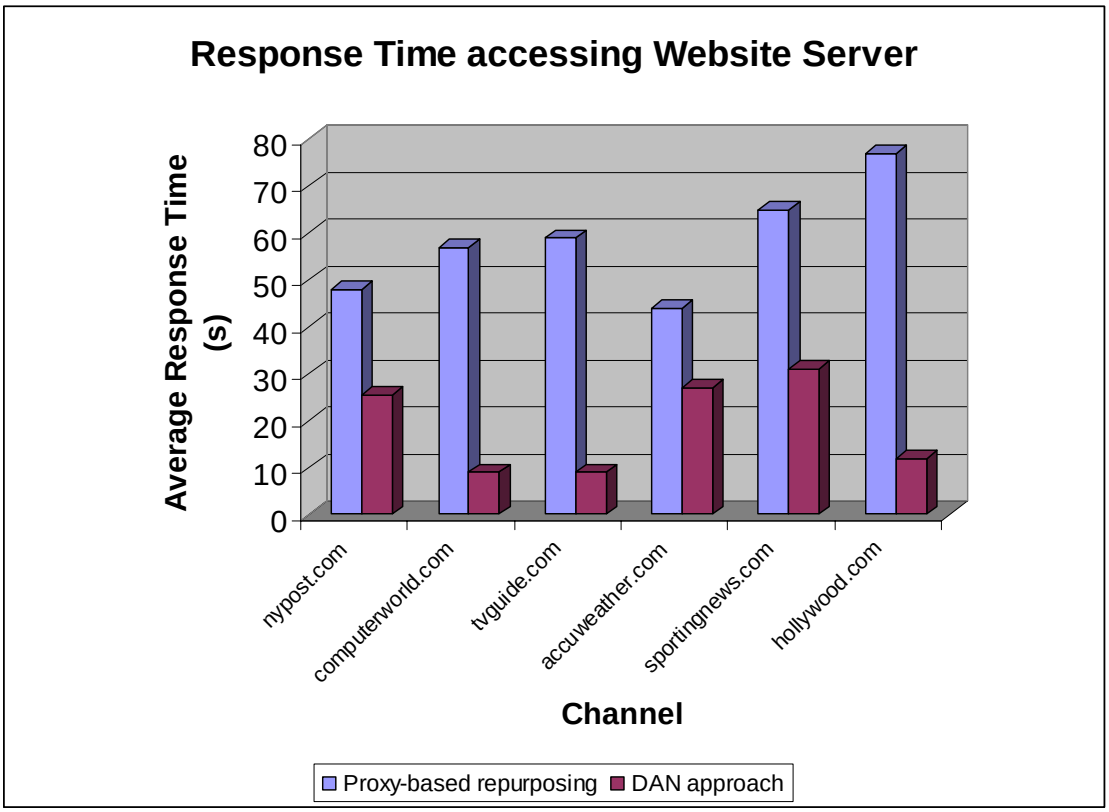
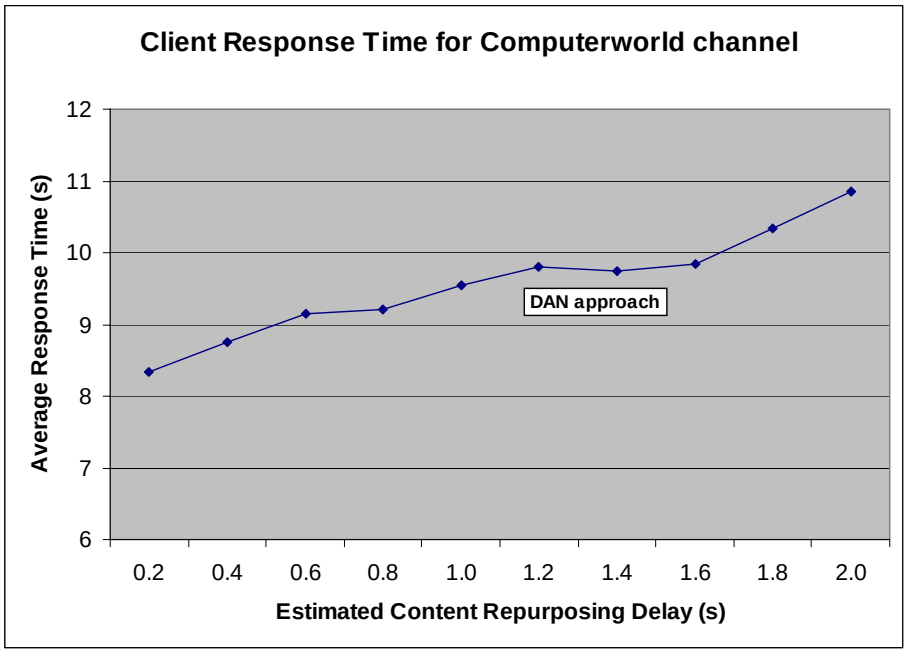


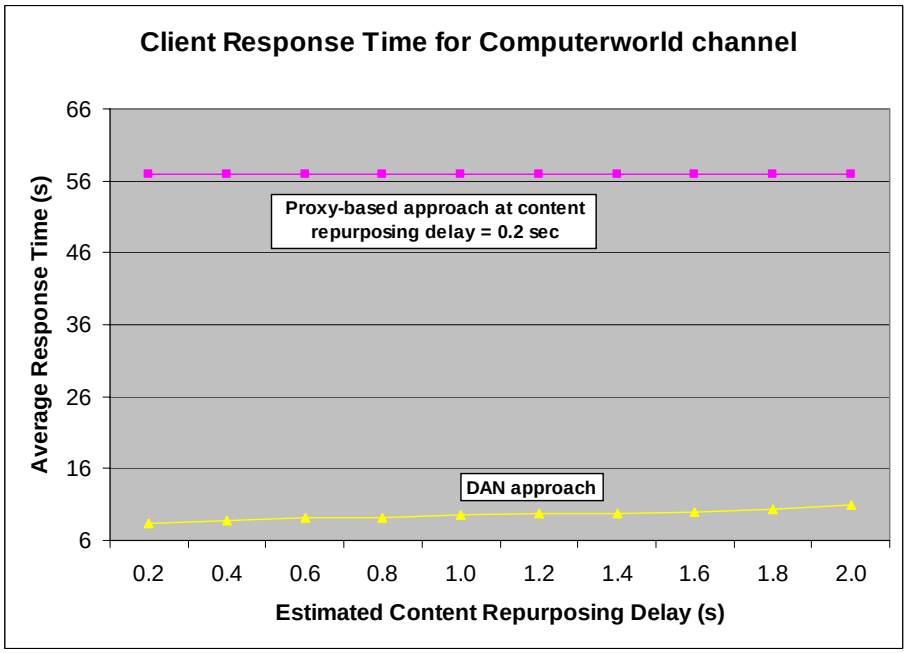
Figure 42. Comparing response time between proxy-based repurposing and DAN approach

As a result of network bandwidth conservation, the response times experienced by clients accessing a typical webpage from the website servers using DAN approach are also significantly better than that for proxy-based approach (Figure 42). The highest improvement occurs at Hollywood channel with a reduction of 65 s in response time, and the minimum improvement occurs at Accuweather channel with reduction of only 17 s in response time. These results illustrated the advantages of DAN approach in optimizing network bandwidth usage, and thus leading to better overall user experience.

In addition, the content repurposing processing delay at the DAN-enabled router is varied from 0.2 s to 2.0 s for Computerworld channel using DAN approach, and the effect on the performance of response time is simulated and shown in the figure below:



(a) For DAN approach only



(b) Comparing DAN with proxy-based approach

Figure 43. Effect on response time with varying content repurposing delay

As observed in Figure 43(a), the average response time has a linear relationship with content repurposing processing delay that occurs at DAN-enable router, in the DAN approach. Comparing with proxy-based approach at content repurposing delay of 0.2 s (Figure 43(b)), DAN approach is still significantly better in performance even when the content repurposing delay is increased to 2.0 s.

E. SIMULATION MODEL C: DEVICE PROFILE ENCAPSULATION

This model is developed using OMNeT++ simulation software, to analyze the impact of utilizing IP Option field in IP header to encapsulate device profile information. Response times of clients accessing a content server are recorded to study the impact.

1. Network Description (NED) File

In OMNeT++, the Network Description (NED) language describes the network environment. The NED description file will enable the simulation model to automatically build the network environment based on the parameters supplied by the designer. For this model, the parameters used in defining the environment are shown in the following table:

Parameter	Description
num_clients	The number of clients deployed in the simulated environment. The number will affect the number of connections created on the Router module to support router-client network connections. In the simulation, the number of clients is assumed to be five.
error_rate	Bit error rate of the network connections. It is the probability that a bit is incorrectly transmitted. The message has an error flag which is set in case of transmission errors. In the simulation, the bit error rate is assumed to be 0.00001, typical of a wireless LAN environment.
data_rates	The data rate specified for the network connections. It is used to calculate transmission latency. In the simulation, the data rate is assumed to be 1 Mbps.

Parameter	Description
reply_size	The size of the message replied by the server when a request is sent from the client. In the simulation, the message size is assumed to be 5 KB, based on worst-case scenario since additional header overhead is likely to have more significant effect on smaller message size.
frame_size	The size of the frame used in the medium access control layer. 1518 and 2346 bytes are used for separate scenarios in the simulation, which are the typical frame sizes used in wireless LAN environment.
ip_option	The assumed overhead in bytes used to encapsulate device profile information. Maximum IP Option overhead is 40 bytes.

Table 9. Parameters used in NED description file

The NED description file also defines the simple modules used in the simulation model - client, server and router modules. Each simple module definition includes information on gates and connections used to inter-connect each of the simple modules. For the simulation model developed, the communication channels between the simple modules are defined with a propagation delay of 10 ms, data-rate specified by the parameter “data_rates”, and bit error rate specified by the parameter “error_rate”. A diagram showing how the different modules are connected in the network environment is shown in Figure 44. The NED description file is provided in Appendix B.

2. Implementation Details

The client will request for content from the server periodically by sending a request message of 1 KB in size. A timestamp is recorded in the client node when the request message is sent. Upon receipt of a request message, the server replies with a reply message equal to the size specified in parameter “reply_size”. The packet undergoes a fragmentation process into smaller frames of size specified in parameter “frame_size”,

before transmitting in a continuous stream on the connection towards the router. The connection is prone to errors with a bit error rate specified in parameter “error_rate”.

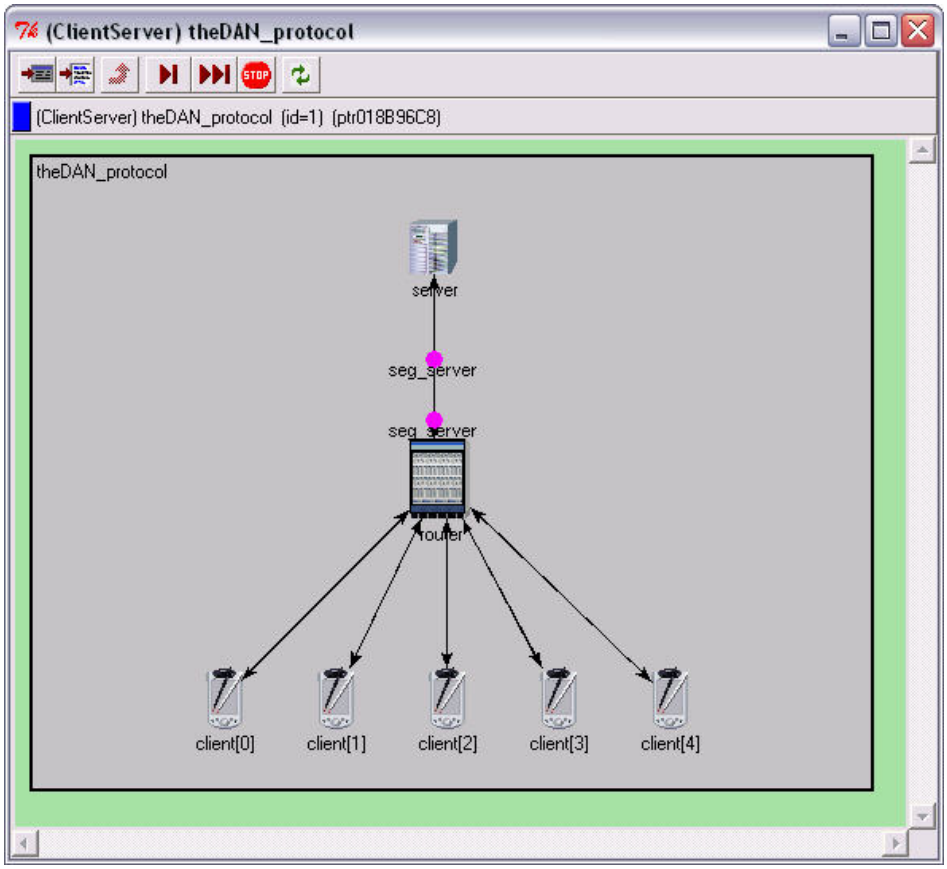


Figure 44. OMNeT++ simulation model

As the frames arrive separately in the client, they are re-assembled into the entire message before the next request is sent. When one (or more) of the frames belonging to the same message is detected with errors, an “NACK” message is sent to the server to request for re-transmission of the entire message. Likewise, when a request is detected with error by the server, an “NACK” message is sent to request for re-transmission from the client. When the message is successfully re-assembled at the client, a second timestamp is recorded, and the difference is the response time. The response time is dependent on the following:

- Network transmission time. It is the round-trip transmission time, which is dependent on the message size and bandwidth (specified in parameter “data_rates”).

- Network propagation delay specified in NED description file when defining the communication channels.
- Processing delay in the client, server and router.
- Probability of bit error specified in parameter “error_rate”, which leads to re-transmission of the message.

The C++ source codes for the client, server and router modules are provided in Appendix B.

3. Running the Simulation

Four scenarios are run using the simulation model, and the differences in parameter values used in “frame_size” and “ip_option” are shown in Table 10. The simulations are run over a period of 1000 s specified in the omnetpp.ini file.

Scenario	Frame size (in bytes)	IP Option overhead (in bytes)
Scenario 1	1518	0
Scenario 2	1518	40
Scenario 3	2346	0
Scenario 4	2346	40

Table 10. Description of scenarios used in OMNeT++ model

F. COMPARING SIMULATION RESULTS FOR SIMULATION MODEL C

Over the simulation runs, response times for the different scenarios are consolidated, and different statistical results of the response times are computed and tabulated, as shown in Table 11, and also shown graphically in Figure 45.

Response Time (in seconds)	1518-byte frame size with No IP Option	1518-byte frame size with 40-byte IP Option	2346-byte frame size with No IP Option	2346-byte frame size with 40-byte IP Option
Average	2.717269753	2.931723878	2.091063087	2.08237693
Maximum	12.539366	13.0125248	8.303797482	8.91024156
Minimum	0.889150697	0.865455601	0.728720525	0.732544353
Standard Deviation	2.070111846	2.367644508	1.516164458	1.544336776

Table 11. Computation of client response times

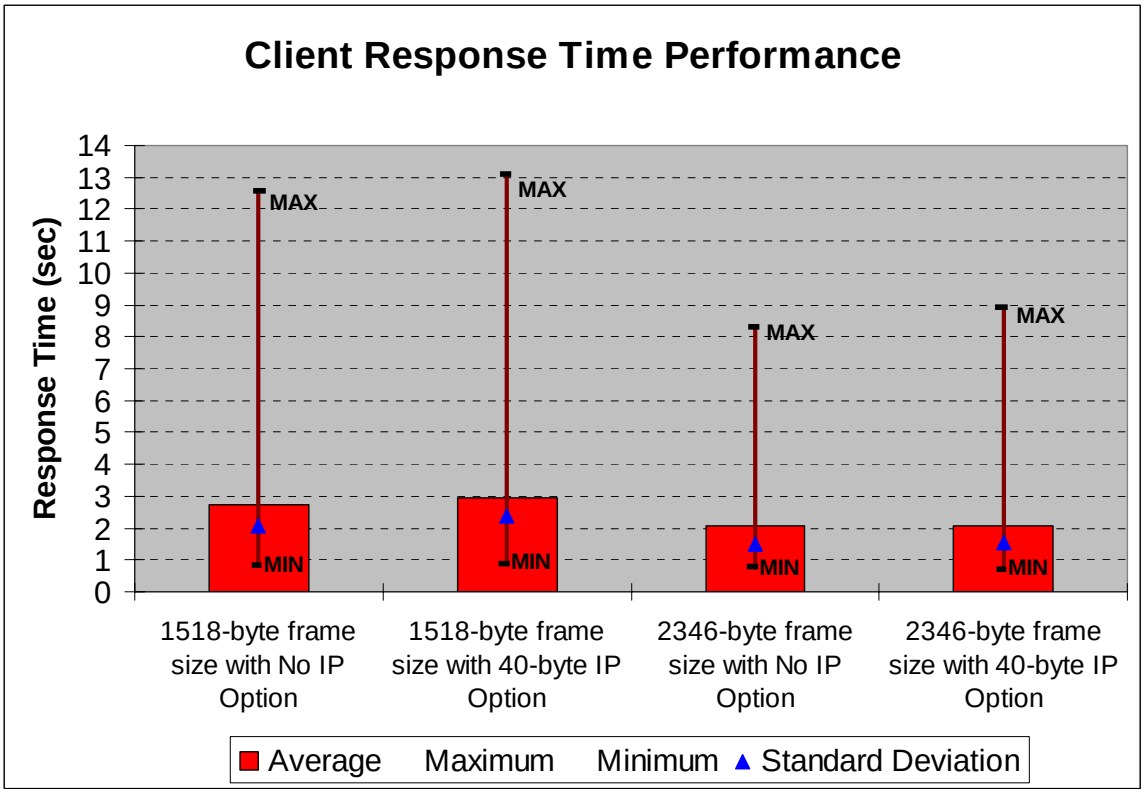


Figure 45. Graphical results of client response times

At both 1518 and 2346-byte frame sizes, even with the addition of maximum IP Option at 40 bytes, the performance in response time is not affected significantly. Thus, the proposed encapsulation of device profile information using IP Option field in the IP header for DAN network will not affect the overall performance of the system. The large variations in the maximum and minimum values for the response times are due to the re-transmission of messages when a bit error is detected in the transmission. The bit error rate set for this case is 0.00001. The variation will reduce as the bit error rate is decreased.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

As the popularity of Internet soars, the content on the Internet is increasingly accessed by wireless and mobile access devices that are usually small in form factor and limited in resources, such as smaller display screen; lesser processing power, memory and battery power; and connected to the Internet with limited bandwidth. Because content in the Internet varies in types and serves many different purposes, the need to repurpose the content to fit the device capabilities becomes an important task in the development of Internet. By enabling device-awareness in the network, unnecessary wastage of network and device resources can be avoided, and only "usable" content is delivered to the end device.

This report has presented a review of existing content repurposing frameworks and their limitations. In contrast to these limitations, a more efficient approach to enable device-aware networking has been proposed in this report, which encapsulates necessary device profile information in transmitting packets and incorporating content repurposing functionality in existing network entities along the data path, preferably closest to the content sources.

Simulation models are developed to statistically evaluate the performance of the proposed DAN architecture in comparison to existing content repurposing frameworks. The simulation results showed that the proposed DAN approach provides a faster framework than typical proxy-based approach. This is the result of intercepting the large content early in transition, and formatting the content into a format suitable for the resource-limited wireless and mobile devices, so that network bandwidth usage is optimized and response time is substantially reduced.

The simulation results also showed that using IP Option field in IP header for encapsulation of device profile information will not have any significant impact in the performance of the system. The simulations were conducted assuming a worst-case scenario of a wireless environment, subjected to transmission errors, and a small reply message size.

Although the simulation models demonstrated the feasibility and suitability of DAN architecture in providing the infrastructure necessary for device capability and content compatibility matching, much work remains to be done for realization of DAN. For example, modifications need to be made to end systems to encapsulate device profile information, and also to extract the information for compatibility analysis. Also, other relevant parameters and algorithms need to be explored and investigated to improve the efficiency of the compatibility policy in decision-making on the need and extend of content repurposing. These future works in DAN will prove important and improve the overall user experience as people adopt the idea of accessing the Internet on the move with wireless and mobile devices.

APPENDIX A – USER AGENT PROFILE FOR NOKIA 6650

The profile can be accessed at http://nds1.nds.nokia.com/uaprof/N6650r300.xml [Accessed October, 2004].

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:prf="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-
20021212#" xmlns:mms="http://www.wapforum.org/profiles/MMS/ccppschema-
20010111#">
 <rdf:Description rdf:ID="Nokia6650">
 <prf:component>
 <rdf:Description rdf:ID="HardwarePlatform">
  <rdf:type
rdf:resource="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-
20021212#HardwarePlatform" />
  <prf:ScreenSize>128x115</prf:ScreenSize>
  <prf:Model>6650</prf:Model>
 <prf:InputCharSet>
 <rdf:Bag>
  <rdf:li>US-ASCII</rdf:li>
  <rdf:li>UTF-8</rdf:li>
  <rdf:li>ISO-8859-1</rdf:li>
  <rdf:li>ISO-10646-UCS-2</rdf:li>
  </rdf:Bag>
  </prf:InputCharSet>
  <prf:ScreenSizeChar>25x7</prf:ScreenSizeChar>
  <prf:BitsPerPixel>12</prf:BitsPerPixel>
  <prf:ColorCapable>Yes</prf:ColorCapable>
  <prf:TextInputCapable>Yes</prf:TextInputCapable>
  <prf:ImageCapable>Yes</prf:ImageCapable>
  <prf:Keyboard>PhoneKeypad</prf:Keyboard>
  <prf:NumberOfSoftKeys>3</prf:NumberOfSoftKeys>
  <prf:Vendor>Nokia</prf:Vendor>
 <prf:OutputCharSet>
 <rdf:Bag>
  <rdf:li>ISO-8859-1</rdf:li>
  <rdf:li>US-ASCII</rdf:li>
  <rdf:li>UTF-8</rdf:li>
  <rdf:li>ISO-10646-UCS-2</rdf:li>
  </rdf:Bag>
  </prf:OutputCharSet>
  <prf:SoundOutputCapable>Yes</prf:SoundOutputCapable>
  <prf:StandardFontProportional>Yes</prf:StandardFontProportional>
```

<prf:PixelAspectRatio>1x1</prf:PixelAspectRatio>
<prf:VoiceInputCapable>Yes</prf:VoiceInputCapable>
</rdf:Description>
</prf:component>
<prf:component>
<rdf:Description rdf:ID="SoftwarePlatform">
<rdf:type
rdf:resource="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-20021212#SoftwarePlatform" />
<prf:AcceptDownloadableSoftware>Yes</prf:AcceptDownloadableSoftware>
<prf:AudioInputEncoder>
<rdf:Bag>
<rdf:li>amr</rdf:li>
</rdf:Bag>
</prf:AudioInputEncoder>
<prf:JavaEnabled>Yes</prf:JavaEnabled>
<prf:JavaPlatform>
<rdf:Bag>
<rdf:li>MIDP/1.0</rdf:li>
<rdf:li>CLDC/1.0</rdf:li>
</rdf:Bag>
</prf:JavaPlatform>
<prf:JavaProtocol>
<rdf:Bag>
<rdf:li>http</rdf:li>
<rdf:li>sms</rdf:li>
<rdf:li>socket</rdf:li>
</rdf:Bag>
</prf:JavaProtocol>
<prf:DownloadableSoftwareSupport>
<rdf:Bag>
<rdf:li>application/vnd.sun.java</rdf:li>
<rdf:li>text/vnd.sun.j2me.app-descriptor</rdf:li>
<rdf:li>application/java-archive</rdf:li>
</rdf:Bag>
</prf:DownloadableSoftwareSupport>
<prf:CcppAccept>
<rdf:Bag>
<rdf:li>application/vnd.wap.wmlscriptc</rdf:li>
<rdf:li>application/vnd.wap.wmlc</rdf:li>
<rdf:li>application/vnd.wap.xhtml+xml</rdf:li>
<rdf:li>application/vnd.wap.sic</rdf:li>
<rdf:li>application/vnd.wap.slc</rdf:li>
<rdf:li>application/vnd.wap.hashed-certificate</rdf:li>
<rdf:li>application/vnd.wap.signed-certificate</rdf:li>
<rdf:li>application/vnd.wap.connectivity-wbxml</rdf:li>

<rdf:li>application/vnd.oma.drm.message</rdf:li>
<rdf:li>application/vnd.oma.drm.rights+xml</rdf:li>
<rdf:li>application/vnd.oma.drm.rights+wbxml</rdf:li>
<rdf:li>application/vnd.oma.drm.content</rdf:li>
<rdf:li>application/vnd.oma.dd+xml</rdf:li>
<rdf:li>application/vnd.nokia.ringing-tone</rdf:li>
<rdf:li>application/java</rdf:li>
<rdf:li>application/java-archive</rdf:li>
<rdf:li>application/x-java-archive</rdf:li>
<rdf:li>application/x-wap-prov.browser-bookmarks</rdf:li>
<rdf:li>application/vnd.wap.cert-response</rdf:li>
<rdf:li>application/xhtml+xml</rdf:li>
<rdf:li>application/x-wallet-appl.user-data-provision</rdf:li>
<rdf:li>application/vnd.met.receipt</rdf:li>
<rdf:li>audio/amr</rdf:li>
<rdf:li>audio/mid</rdf:li>
<rdf:li>audio/midi</rdf:li>
<rdf:li>audio/x-mid</rdf:li>
<rdf:li>audio/x-midi</rdf:li>
<rdf:li>audio/sp-midi</rdf:li>
<rdf:li>audio/3gpp</rdf:li>
<rdf:li>audio/mp4</rdf:li>
<rdf:li>video/3gpp</rdf:li>
<rdf:li>video/mp4</rdf:li>
<rdf:li>text/x-vCard</rdf:li>
<rdf:li>text/x-vCalendar</rdf:li>
<rdf:li>text/vnd.wap.wml</rdf:li>
<rdf:li>text/vnd.wap.wmlscript</rdf:li>
<rdf:li>text/x-co-desc</rdf:li>
<rdf:li>text/css</rdf:li>
<rdf:li>text/html</rdf:li>
<rdf:li>text/vnd.wap.sl</rdf:li>
<rdf:li>text/vnd.wap.si</rdf:li>
<rdf:li>text/vnd.sun.j2me.app-descriptor</rdf:li>
<rdf:li>image/vnd.wap.wbmp</rdf:li>
<rdf:li>image/jpeg</rdf:li>
<rdf:li>image/jpg</rdf:li>
<rdf:li>image/bmp</rdf:li>
<rdf:li>image/gif</rdf:li>
<rdf:li>image/png</rdf:li>
<rdf:li>image/vnd.nok-oplogo-color</rdf:li>
<rdf:li>image/vnd.nok-wallpaper</rdf:li>
</rdf:Bag>
</prf:CcppAccept>
<prf:CcppAccept-Charset>
<rdf:Bag>

<rdf:li>US-ASCII</rdf:li>
<rdf:li>ISO-8859-1</rdf:li>
<rdf:li>UTF-8</rdf:li>
<rdf:li>ISO-10646-UCS-2</rdf:li>
</rdf:Bag>
</prf:CcppAccept-Charset>
<prf:CcppAccept-Encoding>
<rdf:Bag>
<rdf:li>base64</rdf:li>
</rdf:Bag>
</prf:CcppAccept-Encoding>
</rdf:Description>
</prf:component>
<prf:component>
<rdf:Description rdf:ID="NetworkCharacteristics">
<rdf:type
rdf:resource="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-
20021212#NetworkCharacteristics" />
<prf:SecuritySupport>
<rdf:Bag>
<rdf:li>signText</rdf:li>
<rdf:li>TLS</rdf:li>
<rdf:li>SSL</rdf:li>
</rdf:Bag>
</prf:SecuritySupport>
<prf:SupportedBearers>
<rdf:Bag>
<rdf:li>GPRS</rdf:li>
</rdf:Bag>
</prf:SupportedBearers>
</rdf:Description>
</prf:component>
<prf:component>
<rdf:Description rdf:ID="BrowserUA">
<rdf:type
rdf:resource="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-
20021212#BrowserUA" />
<prf:BrowserName>Nokia</prf:BrowserName>
<prf:DownloadableBrowserApps>
<rdf:Bag>
<rdf:li>application/vnd.nokia.ringing-tone</rdf:li>
</rdf:Bag>
</prf:DownloadableBrowserApps>
<prf:JavaScriptEnabled>No</prf:JavaScriptEnabled>
<prf:FramesCapable>No</prf:FramesCapable>
<prf:TablesCapable>Yes</prf:TablesCapable>

<prf:XhtmlVersion>1.0</prf:XhtmlVersion>
<prf:XhtmlModules>
<rdf:Bag>
<rdf:li>xhtml-basic10</rdf:li>
</rdf:Bag>
</prf:XhtmlModules>
</rdf:Description>
</prf:component>
<prf:component>
<rdf:Description rdf:ID="WapCharacteristics">
<rdf:type
rdf:resource="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-20021212#WapCharacteristics" />
<prf:WapDeviceClass>C</prf:WapDeviceClass>
<prf:WapVersion>2.0</prf:WapVersion>
<prf:WmlVersion>
<rdf:Bag>
<rdf:li>1.3</rdf:li>
</rdf:Bag>
</prf:WmlVersion>
<prf:WmlDeckSize>51200</prf:WmlDeckSize>
<prf:WmlScriptVersion>
<rdf:Bag>
<rdf:li>1.2</rdf:li>
</rdf:Bag>
</prf:WmlScriptVersion>
<prf:WmlScriptLibraries>
<rdf:Bag>
<rdf:li>Lang</rdf:li>
<rdf:li>Float</rdf:li>
<rdf:li>String</rdf:li>
<rdf:li>URL</rdf:li>
<rdf:li>WMLBrowser</rdf:li>
<rdf:li>Dialogs</rdf:li>
</rdf:Bag>
</prf:WmlScriptLibraries>
<prf:WtaiLibraries>
<rdf:Bag>
<rdf:li>WTA.Public.makeCall</rdf:li>
<rdf:li>WTA.Public.sendDTMF</rdf:li>
<rdf:li>WTA.Public.addPBEntry</rdf:li>
</rdf:Bag>
</prf:WtaiLibraries>
<prf:DrmClass>
<rdf:Bag>
<rdf:li>ForwardLock</rdf:li>

<rdf:li>CombinedDelivery</rdf:li>
<rdf:li>SeparateDelivery</rdf:li>
</rdf:Bag>
</prf:DrmClass>
<prf:OmaDownload>Yes</prf:OmaDownload>
</rdf:Description>
</prf:component>
<prf:component>
<rdf:Description rdf:ID="PushCharacteristics">
<rdf:type
rdf:resource="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema-
20021212#PushCharacteristics" />
<prf:Push-Accept>
<rdf:Bag>
<rdf:li>application/wml+xml</rdf:li>
<rdf:li>text/html</rdf:li>
</rdf:Bag>
</prf:Push-Accept>
<prf:Push-Accept-Charset>
<rdf:Bag>
<rdf:li>US-ASCII</rdf:li>
<rdf:li>ISO-8859-1</rdf:li>
<rdf:li>UTF-8</rdf:li>
<rdf:li>ISO-10646-UCS-2</rdf:li>
</rdf:Bag>
</prf:Push-Accept-Charset>
<prf:Push-Accept-Encoding>
<rdf:Bag>
<rdf:li>base64</rdf:li>
<rdf:li>quoted-printable</rdf:li>
</rdf:Bag>
</prf:Push-Accept-Encoding>
<prf:Push-Accept-AppID>
<rdf:Bag>
<rdf:li>x-wap-application:wml.ua</rdf:li>
<rdf:li>*</rdf:li>
</rdf:Bag>
</prf:Push-Accept-AppID>
<prf:Push-MsgSize>1400</prf:Push-MsgSize>
</rdf:Description>
</prf:component>
<prf:component>
<rdf:Description rdf:ID="MmsCharacteristics">
<rdf:type rdf:resource="http://www.wapforum.org/profiles/MMS/ccppschema-
20010111#MmsCharacteristics" />
<mms:MmsMaxMessageSize>102400</mms:MmsMaxMessageSize>

```
 <mms:MmsMaxImageResolution>640x480</mms:MmsMaxImageResolution>
<mms:MmsCcppAccept>
<rdf:Bag>
 <rdf:li>application/vnd.nokia.ringing-tone</rdf:li>
 <rdf:li>application/vnd.oma.drm.message</rdf:li>
 <rdf:li>audio/mid</rdf:li>
 <rdf:li>audio/midi</rdf:li>
 <rdf:li>audio/x-mid</rdf:li>
 <rdf:li>audio/x-midi</rdf:li>
 <rdf:li>audio/sp-midi</rdf:li>
 <rdf:li>audio/amr</rdf:li>
 <rdf:li>audio/amr-wb</rdf:li>
 <rdf:li>image/jpg</rdf:li>
 <rdf:li>image/jpeg</rdf:li>
 <rdf:li>image/gif</rdf:li>
 <rdf:li>image/png</rdf:li>
 <rdf:li>image/bmp</rdf:li>
 <rdf:li>image/vnd.wap.wbmp</rdf:li>
 <rdf:li>text/x-vCard</rdf:li>
 <rdf:li>text/x-vCalendar</rdf:li>
 <rdf:li>text/plain</rdf:li>
 <rdf:li>video/3gpp</rdf:li>
 </rdf:Bag>
 </mms:MmsCcppAccept>
<mms:MmsCcppAcceptCharSet>
<rdf:Bag>
 <rdf:li>UTF-8</rdf:li>
 <rdf:li>ISO-8859-1</rdf:li>
 <rdf:li>US-ASCII</rdf:li>
 <rdf:li>ISO-10646-UCS-2</rdf:li>
 </rdf:Bag>
 </mms:MmsCcppAcceptCharSet>
<mms:MmsVersion>
<rdf:Bag>
 <rdf:li>1.0</rdf:li>
 </rdf:Bag>
 </mms:MmsVersion>
 </rdf:Description>
 </prf:component>
 </rdf:Description>
</rdf:RDF>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B – SOURCE CODES FOR OMNET++ MODEL

A. DAN_PROTOCOL.NED

```
//---------------------------------------------------------------
// file: DAN_protocol.ned
//---------------------------------------------------------------

// Client --
//
// A client PDA which periodically connects to the server for data exchange.
//
simple Client
    gates:
        out: out;
        in: in;
endsimple

// Router --
//
// A very simple module which models the network component (DAN router) between
// the server and the clients, which has the capability to perform compatibility processing
// and content repurposing.
//
simple Router
    gates:
        out: out[];
        in: in[];
endsimple

// Server --
//
// Models a simple server which accepts connections from the client PDAs. It serves
// multiple connections at a time; each connection is handled by a ServerProcess module,
// created on demand.
//
simple Server
    gates:
        out: out;
        in: in;
endsimple

// ClientServer --
//
// Model of the network, consisting of serveral clients, a server and a router.
//
```

```
module ClientServer
   parameters:
      num_clients : numeric,
      error_rate : numeric,
      data_rates : numeric,
      req_size : numeric,
      reply_size : numeric,
      frame_size : numeric,
      ip_option : numeric;
   submodules:
      server: Server;
         display: "p=263,67;i=server1;b=30,34";
      router: Router;
         gatesizes:
            in[num_clients+1],
            out[num_clients+1];
         display: "p=267,212;i=router3;b=38,50";
      client: Client[num_clients];
         display: "p=131,349,r,70;i=pda3;b=23,39";
   connections:
      for i=0..num_clients-1 do
         client[i].out --> delay 10ms --> router.in[i];
         client[i].in <-- delay 10ms <-- router.out[i];
      endfor;
      server.out --> delay 10ms datarate data_rates error error_rate -->
router.in[num_clients];
      server.in <-- delay 10ms datarate data_rates error error_rate <--
router.out[num_clients];
   display: "p=10,10;b=529,397";
endmodule

// theDAN_protocol --
//
// Instantiates a ClientServer network.
//
network theDAN_protocol : ClientServer
   parameters:
      num_clients = input(5,"Number of clients:"),
      error_rate = input(0.00001, "Network Error Rate:"),
      frame_size = input(1518, "Size of Frame (bytes):"),
      req_size = 1024*8, // 1KB
      reply_size = input(5, "Reply Size (KB):"),
      ip_option = input(0, "IP Option Field (bytes):"),
      data_rates = input(1000000,"Data Rate (bps):");
endnetwork
```

B. CLIENT.CPP

```
//--------------------------------------------------------------
// File: client.cc
// Modified from part of DYNA - an OMNeT++ demo simulation
//--------------------------------------------------------------

#include "omnetpp.h"

class Client : public cSimpleModule
{
   Module_Class_Members(Client,cSimpleModule,16384)
   virtual void activity();
};

Define_Module( Client );

void Client::activity()
{
  // variable declaration
  int own_addr = gate( "out" )->toGate()->index();
  int server_addr = gate( "out" )->toGate()->size()-1;
  int req_size = parentModule()->par("req_size");
  int reply_size = parentModule()->par("reply_size");
  int frame_size = parentModule()->par("frame_size");
  int ip_option = parentModule()->par("ip_option");
  reply_size = reply_size*1024*8 + 160 + (ip_option*8);  // include encapsulation of
device profile
  int num_packet = reply_size/(frame_size*8);
  int remain_size = reply_size - (frame_size*8)*num_packet;
  int counter, flag, count;
  double response_time;
  bool req_retransmit;
  cOutVector resp_v("response_time");
  cMessage *done;

  // to determine the number of frames expected to receive from server
  if (remain_size != 0)
          counter = num_packet+1;
  else
          counter = num_packet;

    for(;;)
    {
       // keep an interval between subsequent connections
       wait( uniform(3.0,15.0) );
```

```
		// connection setup. Generating and sending request including device
profile to server
	ev << "Client " << name() << " sending request and device profile\n";
	cMessage *work = new cMessage( name() );
	work->addPar("src") = own_addr;
	work->addPar("dest") = server_addr;
		work->setLength(req_size+160+ip_option*8); // include encapsulation of
device profile
		work->setKind(0);
		response_time = simTime();
	send( work, "out" );

		// receive reply from server
	ev << "Client " << name() << " waiting for Reply\n";
	done = receive();
		flag = done->par("flag");

		count = 0;
		req_retransmit = false;

		// to handle the case when a negative acknowledgement is received,
	  // indicating the client has send a request with error, or the case whereby it
	  // is still not the last frame received from server
		while ((done->length() == 10) || count < counter)
		{
			if (done->length() == 10)  // NACK received, whether error or not
			{
				ev << "Client " << name() << " sent a bad message!!!\n";
				ev << "Re-send request\n";
				cMessage *work1 = new cMessage( name() );
				work1->addPar("src") = own_addr;
				work1->addPar("dest") = server_addr;
				work1->setLength(req_size+160+ip_option*8);  // include
encapsulation of device profile
				work1->setKind(1);
				send( work1, "out" );
				count = 0;	// reset count
				ev << "Client " << name() << " waiting for Reply (E)\n";
				delete done;
				done = receive();
				flag = done->par("flag");
			}

			else if (flag == 0 && done->hasBitError())   // to differentiate from
NACK with error
			{
```

```
                        req_retransmit = true;
                        count ++;
                        delete done;
                        done = receive();
                        flag = done->par("flag");
                }

                else if ((flag == 1 && done->hasBitError()) || (flag == 1 &&
req_retransmit == true))  // send NACK
                {
                        ev << "Client " << name() << " got a bad message!!!\n";
                        cMessage *work2 = new cMessage( "NACK" );
                        work2->addPar("src") = own_addr;
                        work2->addPar("dest") = server_addr;
                        work2->setLength(10);  // Nack size
                        work2->setKind(2);
                        send( work2, "out" );
                        count = 0;       // reset count
                        ev << "Client " << name() << " waiting for Reply (E)\n";
                        delete done;
                        req_retransmit = false;
                        done = receive();
                        flag = done->par("flag");
                }

                else if (flag == 1) // last frame received
                        count ++;

                else
                {       count ++;
                        delete done;
                        done = receive();
                        flag = done->par("flag");
                }
        }
        // to compute client response time
    response_time = simTime() - response_time;
        resp_v.record(response_time);
    delete done;
  }
}
```

C. SERVER.CPP

```
//---------------------------------------------------------
// File: server.cc
// Modified from part of DYNA - an OMNeT++ demo simulation
```

```
//-------------------------------------------------------------

#include "omnetpp.h"

class Server : public cSimpleModule
{
   Module_Class_Members(Server,cSimpleModule,16384)
   virtual void activity();
};

Define_Module( Server );

void Server::activity()
{
        // variable declaration
   double avg_utilization = 0.0;
        double total_process_time, process_time;
        int reply_size = parentModule()->par("reply_size");
        int frame_size = parentModule()->par("frame_size");
        int ip_option = parentModule()->par("ip_option");
        reply_size = reply_size*1024*8 + 160 + (ip_option*8);  // include encapsulation
of device profile
        int num_packet = reply_size/(frame_size*8);
        int remain_size = reply_size - (frame_size*8)*num_packet;
        cOutVector resp_v("CPU utilization");
        cMessage *msg;

   for(;;)
   {
                total_process_time = 0.0;
      msg = receive();

                // to handle the case when the request received from client has error, or the
                // case whereby negative acknowledgement is received from client,
        // indicating that the server has sent frames with error
                while (msg->hasBitError() || (msg->length() == 10))
                {
                        // server processing delay
                        process_time = uniform(0.05,0.1);
                        wait(process_time);
                        total_process_time = total_process_time + process_time;

                        // generating reply message
                        cMessage *packetr = new cMessage("seg_server");
                        packetr->addPar("src") = msg->par("dest");
                        packetr->addPar("dest")= msg->par("src");
```

```
packetr->addPar("flag") = 0;   // to represent "NOT" the last packet

cMessage *last_packetr = new cMessage ( "seg_server" );
last_packetr->addPar("src") = msg->par("dest");
last_packetr->addPar("dest")= msg->par("src");
last_packetr->addPar("flag") = 1;   // to represent last packet

if (msg->length() == 10)  // Nack message received
{
        packetr->setLength(frame_size*8);
        packetr->setKind(6);
        last_packetr->setKind(7);
        if (remain_size != 0)
                last_packetr->setLength(remain_size);
        else
                last_packetr->setLength(frame_size*8);

        // packetize the message for sending
        for (int i = 0; i < num_packet-1; i++)
        {
                cMessage *copy = (cMessage *) packetr -> dup();
                send( copy, "out" );
        }
        if (remain_size != 0)
        {
                cMessage *copy = (cMessage *) packetr -> dup();
                send( copy, "out" );
                send ( last_packetr, "out");
        }
        else
        {
                send ( last_packetr, "out");
        }

}
else  // Send Nack
{
         cMessage *send_nack = new cMessage("NACK");
         send_nack->addPar("src") = msg->par("dest");
         send_nack->addPar("dest")= msg->par("src");
         send_nack->addPar("flag") = 2; // to represent other packet
         send_nack->setLength(10);
         send_nack->setKind(8);
         send(send_nack, "out");
}
```

```
        delete msg;
        msg = receive();
}

// server processing delay
process_time = uniform(0.05,0.1);
wait(process_time);
total_process_time = total_process_time + process_time;

// generating and sending reply to client
cMessage *packet = new cMessage("seg_server");
packet->addPar("src") = msg->par("dest");
packet->addPar("dest")= msg->par("src");
packet->addPar("flag") = 0;     // to represent "NOT" the last packet
packet->setLength(frame_size*8);
packet->setKind(4);

cMessage *last_packet = new cMessage ( "seg_server" );
last_packet->addPar("src") = msg->par("dest");
last_packet->addPar("dest")= msg->par("src");
last_packet->addPar("flag") = 1;    // to represent last packet
last_packet->setKind(5);

if (remain_size != 0)
        last_packet->setLength(remain_size);
else
        last_packet->setLength(frame_size*8);

delete msg;

// to compute CPU utilization
avg_utilization = avg_utilization + total_process_time;
resp_v.record(avg_utilization/simTime());

// packetize the message for sending
for (int i = 0; i < num_packet-1; i++)
{
        cMessage *copy = (cMessage *) packet -> dup();
        send( copy, "out" );
}
if (remain_size != 0)
{
        cMessage *copy = (cMessage *) packet -> dup();
        send( copy, "out" );
        send ( last_packet, "out");
}
```

```
                else
                {
                        send ( last_packet, "out");
                }
    }
}
```

D. ROUTER.CPP

```
//---------------------------------------------------------------
// File: router.cc
// Modified from part of DYNA - an OMNeT++ demo simulation
//---------------------------------------------------------------

#include "omnetpp.h"

class Router : public cSimpleModule
{
   Module_Class_Members(Router,cSimpleModule,16384)
   virtual void activity();
};

Define_Module( Router );

void Router::activity()
{ // variable declaration
  double avg_utilization = 0.0;
  double process_time;
  double network_util;
  int data_rates = parentModule()->par("data_rates");
  int req_size = parentModule()->par("req_size");
  int reply_size = parentModule()->par("reply_size");
  int server_add = parentModule()->par("num_clients");
  long total_bits = 0;
  cOutVector resp_v("Router utilization");
  cOutVector resp_n("Network utilization");

   for(;;)
   {
      // receive msg (implicit queueing!)
      cMessage *msg = receive();
                int source = msg->par("src");

                // to compute network utilization
                if (source == server_add)
                {
                  int size = msg->length();
```

```
            total_bits = total_bits + size;
            network_util = total_bits / (simTime() * data_rates);
            if (network_util > 1.0) network_util = 1.0;
            resp_n.record(network_util);
          }

          // capability compatibility processing delay &
          // content repurposing processing delay experienced at DAN router
          process_time = uniform(0.1,0.3);
          wait( process_time );

          // to compute router utilization
          avg_utilization = avg_utilization + process_time;
          resp_v.record(avg_utilization/simTime());

          // forward msg to destination
          // assuming that content is compatible with client
          int dest = msg->par("dest");
          ev << "Relaying msg to addr=" << dest << '\n';
          send( msg, "out", dest);
    }
}
```

LIST OF REFERENCES

[AvantGo, 2004] AvantGo, "AvantGo Channel Developer Guide: Version 2.0". Available from: http://www.ianywhere.com/avantgo/developer/channel_developer/index.html. Accessed October 2004.

[Canali et al, 2003] Claudia Canali, Valeria Cardellini and Riccardo Lancellotti, "Squid-based proxy server for content adaptation", Department of Computer Engineering, University of Roma, January 2003. Available from: http://weblab.ing.unimo.it/papers/tr-2003-03.pdf. Accessed October 2004.

[CC/PP, 2004] W3C Recommendation, "Composite Capability / Preference Profiles (CC/PP): Structure and Vocabularies 1.0", January 2004. Available from: http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115. Accessed October 2004.

[Han et al, 1998] Richard Han, Pravin Bhagwat, Richard Lamaire, Todd Mummert, Veronique Perret and Jim Rubas, "Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing", IBM T.J. Watson Research Center, IEEE Personal Communications, December 1998. Available from: http://ieeexplore.ieee.org/iel4/98/15881/00736473.pdf. Accessed October 2004.

[Hu and Bagga, 2004] Jianying Hu and Amit Bagga, "Categorizing Images in Web Documents", IBM T.J. Watson Research Center and Avaya Labs Research, IEEE Computer Society, 2004. Available from: http:// ieeexplore.ieee.org/iel5/93/28183/01261103.pdf. Accessed October 2004.

[Kurose, 2003] James F. Kurose and Keith W. Ross, "Computer Networking: A Top-Down Approach Featuring the Internet", pages 558-563, 2nd edition, Addison Wesley, 2003

[MediaLab, 2004] MediaLab, "Web Content Adaptation White Paper", August 2004. Available from: http://www.medialab.sonera.fi/workspace/WebContentAdaptationWP.pdf. Accessed October 2004.

[Nokia, 2003] Nokia Forum, "Introduction to User Agent Profile", August 2003. Available from: http://nds1.forum.nokia.com/nnds/ForumDownloadServlet?id=3498&name=Introduction_to_User_Agent_Profile_v1_1_en.pdf. Accessed October 2004.

[Opera, 2004] Opera press releases, "#1 for full mobile Web browsing: Opera Mobile Reaches One Million Downloads", September 2004. Available from: http://www.opera.com/pressreleases/en/2004/09/20. Accessed October 2004.

[OPNET, 2004] OPNET Technologies, “OPNET Software Product Documentation”.

[RFC791, 1981] Information Sciences Institute, “RFC 791: Internet Protocol”, September 1981. Available from: http://www.ietf.org/rfc/rfc0791.txt. Accessed October 2004.

[RFC2327, 1998] Network Working Group, “Session Description Protocol”, April 1998. Available from: http://www.ietf.org/rfc/rfc2327.txt. Accessed October 2004.

[RFC2543, 1999] Network Working Group, “Session Initiation Protocol”, March 1999. Available from: http://www.ietf.org/rfc/rfc2543.txt. Accessed October 2004.

[SDPng, 2004] MMUSIC Internet-draft, “Session Description and Capability Negotiation”, April 2004. Available from: http://community.roxen.com/developers/idocs/drafts/draft-ietf-mmusic-sdpng-07.txt. Accessed October 2004.

[Singh, 2004] Gurminder Singh, “Content Repurposing”, IEEE Multimedia, 11(1), pages 20-21, January-March 2004

[Tcpipguide, 2004] The TCP/IP Guide, “IP Datagram Options and Option Format”. Available from: http://www.tcpipguide.com/free/t_IPDatagramOptionsandOptionFormat.htm. Accessed October 2004.

[W3C, 2004] W3C Development, “UAProf Profile Repository”. Available from: http://www.w3development.de/rdf/uaprof_repository. Accessed October 2004.

[Varga, 2003] Andras Varga, “OMNeT++ Version 2.3 User Manual”, June 2003. Available from: http://www.omnetpp.org/external/doc/html/usman.php. Accessed October 2004.

[WAG, 2001] Wireless Application Group User Agent Profile Specifications, “Wireless Application Protocol: WAP-248-UAPROF-20011020-a”, October 2001. Available from: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html. Accessed October 2004.

[WASP, 2004] Web Standards Project, “Introduction to Composite Capabilities / Preferences Profile (CC/PP)”. Available from: http://www.webstandards.org/learn/askw3c/feb2004.html. Accessed October 2004.

[WebSphere, 2004] IBM WebSphere Software, “WebSphere Transcoding Publisher Overview”. Available from: http://www-306.ibm.com/software/pervasive/transcoding_publisher. Accessed October 2004.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Gurminder Singh
   Naval Postgraduate School
   Monterey, California

4. Su Wen
   Naval Postgraduate School
   Monterey, California

5. Tan Lai Poh
   Temasek Defense Systems Institute
   Singapore

6. Sim Siew See, Human Resource Department
   Singapore Defence Science and Technology Agency
   Singapore