

HE, YIHUA, M.S., Aug 2002

COMPUTER SCIENCE

FAST INTERCEPT OF PASSING STREAMS FOR HIGH PERFORMANCE FILTER APPLICANCES IN APPLICATION SERVICE NETWORKING (90 pp.)

Director of Thesis: Javed I. Khan

Internet is increasingly being "active". Documents are being dynamically processed before they are served. The location of processing is also dynamic. The work investigates two aspects: (a) how documents can be processed within an overall service model/scenario in any location between the origin server and the user-agent; (b) what type of network software layer in the intercepting machine can expedite intermediate information processing. Random access into processed data is believed to be an important performance criterion in any computation. Envisioning a generalized framework for supporting a wide range of possible content services, the thesis suggests a novel content scoping and indexing based random access mechanism into a passing stream for intercepting filter like appliances on this framework. It also presents an application programming interface for efficient stream editing. The work also presents a user space implementation of the proposed intercepting machine and a performance study of the scheme on this implementation. Even without any kernel level support, the implementation showed about 500-800% speedup over today's content servicing technique in normal conditions. The result suggests such random access can significantly speed up future intercepting applications of the Internet.

**FAST INTERCEPT OF PASSING STREAMS FOR HIGH PERFORMANCE
FILTER APPLICANCES IN APPLICATION SERVICE NETWORKING**

A thesis submitted to
Kent State University in partial
fulfillment of the requirements for the
degree of Master of Science

By

Yihua He

Aug 2002

Thesis written by

Yihua He

B.E., South China University of Technology, 1999

M.S., Kent State University, 2002

Approved by:

_____, Advisor

_____, Chair, Department of Computer Science

_____, Dean, College of Art and Sciences

TABLE OF CONTENT

Approval	ii
Table of Content	iii
List of Figures	vi
List of Tables and Charts	viii
ACKNOWLEDGEMENT	ix
Chapter 1	
Introduction	1
1.1 Overview	1
1.2 Background and Related Work	5
1.3 Layout of the Thesis	8
Chapter 2	
Active Service Distribution and Localization Model	9
2.1 Architecture and Components	9
2.2 Information Components	13
2.3 ASDL Contracting Model	14
2.4 Classification of Active Services	16
2.5 ASDL SCENARIOS	17

2.5.1 EUI Model	17
2.5.2 CPI Model	19
2.5.3 SPI Model	21
Chapter 3	
EDIP Protocol	24
3.1 In Route Application Service	24
3.2 EDIP Indexing Mechanism	25
3.3 EDIP Header Format	26
3.4 EDIP Encapsulation by Serverlets	28
3.5 EDIP Decapsulation and Indexing/Scoping	32
3.6 Application Processing:	36
3.7 An example illustration	42
3.8 Some Sample Plug-in Applications	49
3.8.1 Active Hyperlinking	49
3.8.2 Advertisement Filtering	53
3.8.3 Screen Size Adjustment and Re-layouting	56
Chapter 4	
The Performance	59
4.1 The Environments	59

4.1.1 Software Environment	60
4.1.2 Hardware Environment	61
4.2 Test Application and Sample Used	61
4.3 Performance Test	62
4.3.1 CPU usages for EDIF service:	62
4.3.2 CPU usages for FSF service:	65
4.3.3 CPU Time Comparison among EDIF, FSF and NR Mode	67
4.3.4 Throughput Comparison among EDIF, FSF and NR Mode	69
4.3.5 Average Package Delays	71
4.4 Further Analysis	72
4.4.1 Space Cost	72
4.4.2 Impact on Different Processor Speeds	75
Conclusion	79
REFERENCES	81
GLOSSARY OF TERMS AND ABBREVIATIONS	85
A Sample User Filter Application	87
A Sample Marker Program at Source Side	89

LIST OF FIGURES

Fig 1.1: Example Service with EDIP Header	4
Fig 2.1: ASDL Architecture and Components	10
Fig 2.2: End-user initiates single service	19
Fig 2.3: Content Provider initiates group service	21
Fig 2.4: Service provider initiates group service	23
Fig 3.1: EDIP Header Format	27
Fig 3.2: EDIP Encapsulation	29
Fig 3.3: EDIP Selective Decapsulation System	32
Fig 3.4: Data Manipulate API Operations	37
Fig 3.5: Example of Content Processing with stream-edit API	39
Fig 3.6: An example service	43
Fig 3.7: IP packages encoded with EDIP headers	45
Fig 3.8: A Web Page before adaptation	51
Fig 3.9: A Web Page After Adaptation	52
Fig 3.10: Before Adaptation: A Yahoo web page with commercials	54
Fig 3.11: After Adaptation: An ad-free Yahoo page	55
Fig 3.12: A web page from Yahoo before adjusting the size and relayouting	56

Fig 3.13a: After size adjustment1	57
Fig 3.13b: After size adjustment2	57
Fig 4.1: CPU Time Cost in EDIF Service Mode	64
Fig 4.2: Relative Percentages of CPU Time Cost in EDIF Service Mode	65
Fig 4.3: CPU Time Cost in FSF Service Mode	66
Fig 4.4: Relative Percentages of CPU Time Cost in EDIF Service Mode	67
Fig 4.5: CPU time comparison among EDIF, FSF and NR mode	68
Fig 4.6: Throughput comparison among EDIF, FSF and NR mode	70
Fig 4.7: Average Packages Delays	71
Fig 4.8: Extra Space Percentage Caused by EDIP Headers	73
Fig 4.9: EDIF's Speedup Over FSF Model with Different a Value	78

LIST OF TABLES AND CHARTS

Chart 2.1: Specification methods between different parties	15
Chart 2.2: A list of example services and their modes	17
Table 3.1: Administrative API Subset	37
Table 3.2: Data Manipulate API Subset	38
Table 3.3: The labellist used in the example shown in Fig 3.6	40
Chart 3.1: Actions in EDI-Filtering (EDIF)	47
Chart 3.2: Actions in Full Search Filtering (FSF) Mode	48
Chart 3.3: Different Actions between EDIF and FSF mode	49

ACKNOWLEDGEMENT

I would like to express my deep gratitude and appreciation to Professor Javed I. Khan, who continually guides and supports me for this thesis. He has not only been an amazing advisor for this thesis but has also been of help on academic and personal matters.

I would also like to thank faculties and stuffs in Department of Computer Science, Kent State University. They provided all kinds of valuable help during the time I study here. Without their generous support, I would not be able to complete my MS degree.

I am also thankful to my friends ---classmates, lab mates, and officemates --- who make my stay at Kent State University so enjoyable and memorable.

Last but not the least, special thanks to my parents without whose love, support and encouragement this would not have been possible. I dedicate this thesis to my parents.

CHAPTER 1

INTRODUCTION

Internet is increasingly being “active”. Documents are being dynamically processed before they are served. The location of processing is also dynamic. In this thesis we investigate two aspects: (a) how documents can be processed within an overall service model/scenario in any location between the origin server and user-agent; (b) what type of network software layer in the intercepting machine can facilitate intermediate information processing.

1.1 Overview

With the growing diversity and broadening geographically coverage of Internet population, there is more need for personalized and localized information. Simple server-client service model is no longer satisfying cyber people. The network, which used to be considered only as transmitting media, now becomes more and more “active” between the information sources and destinations. Beginning from cache, proxy and gateways, new services emerge rapidly. Such services include content adaptation, content personalization, location-aware data insertion, security filters, etc. All of these are fundamentally stream interception machines requiring some form of intermediate access

inside transiting traffic's content. A significant percent of the delivered Internet traffic is now 'touched'.

However, most real-life content data carried over network packets are multi-level hierarchically encapsulated and lack of indexing mechanism. Searching and pinpointing exactly where content adaptation should take place are exhausting jobs for conventional filter programs. For example, in annotation based web content transcoding [2], the transcoder has to make considerable effort in searching a certain value in the annotations (basically tags/metadata expressed by XML) at the application level before it can decide how the content should be transcoded. To make things worse, lack of scoping mechanism in IP packet structures, a filter program does not know if a passing-by stream is in its service range, and it has to be decapsulated and encapsulated anyway. For example, in an advertisement insertion service, a filter program should be only interested in HTML streams passing by. However, current existing filtering programs have to decapsulate all passing streams, which may include JPEG, MPEG or MP3 streams that the filter program is not supposed to do anything with.

Lacking the facilities for indexing and scoping for passing-by streams, current protocol design and protocol packet structure are inefficient in dealing with filtering and content adaptation in network. The problem has major implication on the CPU cycle, memory size, and overall performance of any intercepting appliance system's architecture.

The stream is the working data structure of these capsules. It is perhaps as salient to appliance's overall architecture as the design of disk scheduling algorithm or multilevel memory/cache organization is to the conventional machine architecture. Although simple end-to-end applications may do away with marginal treatment of this issue, indeed, we believe right placement of protocol element inside data stream and some form of random access will be one of the most important factor for high performance stream data processing appliances.

Meanwhile, a group of network overlays emerge as frameworks trying to provide systematic execution environments to the increasing Internet services. Though the first generation CDNs emerged as "passive" temporary caching proxies of HTTP responses now we are seeing increased array of other services for customized content delivery that needs "active" computation ability at various intermediate points in the information network. These points will act as the hub for various actions ranging from rich domain knowledge based information steering, filtering, multiplexing, adaptation that will be required by ubiquitous services. Unfortunately, at present there is a serious gap in Internet protocol suit that can provide systematic support for these emerging services. Currently most such 'adaptations' are simulated in content provider's own site typically with arrays of backend servers. Such content servicing is mostly isolated and lacks interoperability or scalability. The overall growth scenario lacks any roadmap for sustained evolution of

such services.

In this thesis we focus on this little explored but important problem and demonstrate a novel content indexing scheme that can facilitate dynamic index based random access into streams and provide performance boost to intercepting filter like appliances. Though conceptually the mechanism can be implemented in layers above IP, we present an IPV6 [5] based protocol called Embedded Data Indexing Protocol (EDIP). It is an IPv6 extension header based content indexing mechanism, which defines how a Content Provider (CP)'s serverlet can add special marks into the data stream, and how an Adaptation Router (AR) can decode those marks from the data streams and gain pattern dependent random access into the elements of required data stream. An example service with EDIP header is show in Fig 1.1.

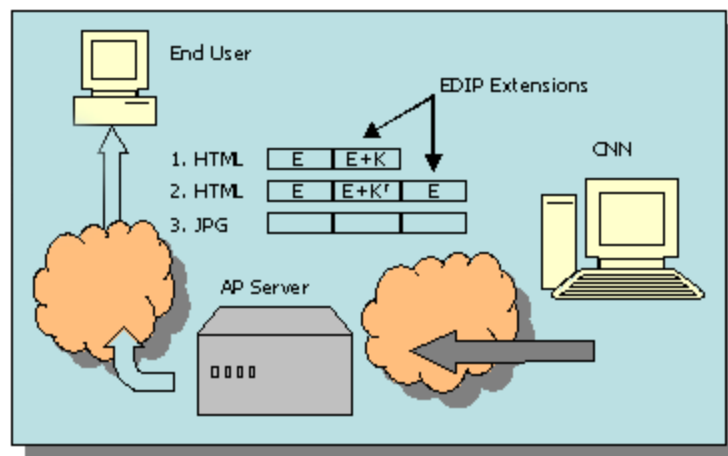


Fig 1.1 Example Service with EDIP Header

We also investigate the vision of a generalized content services network framework,

called Active Service Distribution and Localization (ASDL) model. We outline a framework that can support deployments of wide range of services with various specification and initiation dependencies. However, before presenting the fast filtering mechanism and its framework, first we briefly present some of the very recent and interesting developments in this fast unfolding area.

1.2 Background and Related Work

The first generation of systematic distributed cache coordination began with the proposals for *content distribution networks* (CDN). Commercially, we have seen the emergence of several such content caching systems. The most remarkable one is probably Akamai [3], who provides the global delivery platform for the official web site of the 2002 FIFA World Cup Korea/Japan™ (www.fifaworldcup.com), and making it the most popular sports event in history, according to FIFA¹. Akamai achieves huge success in business by distributing caches/proxy worldwide, which, in concept, provides closer and faster access points to end-users.

Both Akamai and a number of other teams have been looking into technology for content adaptation at an origin server or in those CDN proxy caches. Example works include Spyglass [16], IBM Transcoding proxy [12][17], UC Berkeley TranSend [18], and

¹Akamai's content delivery services and scalable infrastructure have helped support the more than 464 million page views during the first eight days of the tournament (May 31 - June 7, 2002) ---<http://www.akamai.com/en/html/about/customers.html> on Jan 29, 2002.

Mobiware [19]. With provision for value added service in the caching proxies, the IETF Working Group has recently proposed the Open Pluggable Edge Services (OPES) [9][10] and the Internet Content Adaptation Protocol (iCAP) [20] defining the basic functions of future caching proxy. iCAP defined how various caching objects can be transported from one cache to another. OPES provides some interesting capabilities to caching proxies. An OPES proxy can be equipped with message parsers, rule modules, and proxylets library. When messages flow through an OPES proxy, they are not only cached but also automatically parsed and processed with these rules[14]. Ma et. al. [4][6] suggested an enhanced model of *content services networking* (CSN) pursuing a more powerful view of the application server (or proxy). Ma's CSN separates passive caching proxies from application servers. The application servers can directly communication with the content servers and user-agents. Ma shows indeed this approach can handle more service scenarios. These include post or pre distribution services either on behalf of the user agent or on behalf of the content-provider. Also, it allows for more versatile services to be placed into the system as the processing is performed entirely in the application server—a separate entity than the caching proxy.

However, both OPES and basic CSN are still considered overly proxy centric. This approach does not provide enough flexibility in accommodating various service arrangements that may arise in the real service deployments, which often restricts where,

when and how the service can be performed, redirected, and who may provide the service specifications. If we look back into the OPES scenario, we will find that a service provider may download a set of “rules”, and interpret and execute it by the rule processor. Here, rules are actually a set of program language with specific purpose. However, we noticed some limitations in such scenario. An obvious one is that, although the OPES model can be configured to “source-centric” or “client-centric”[10], there is not an easy way for the client to gain help from the source or vice versa. An alternative way to describe the “rules” is by a pair of tightly coupled program, distributed by a single authority to both the service source and the service client. Our ASDL[13] model is such an infrastructure which is considered “service centric”, and EDIP, which is a special IPv6 extension, can be used as a media carrying the helping information in this case.

The idea of putting information in IP level headers is not new, but little effort has been made in utilizing it in value-added services. S. Blake discussed about the differentiated services by adding marker field DS in IPv4 and IPv6 headers [1]. Packets marked by this field will receive a particular per-hop forwarding behavior on nodes along their path. It is a close approach as our EDIP. However, they didn’t investigate the possibility to add indexing information into IP headers and utilize it in value-added service to make random access of the data stream available. Spatscheck [7] and D. Maltz [8] have separately presented two TCP splicing mechanisms which would allow a filter

(connected by two TCP links at two ends), to shed-off some TCP window maintenance functions, for passive filters by splicing the two TCP stream at two end-points.

The technique we propose accelerates the actual filtering operation and applications, as much as it helps the networking layers. Also, the gain is not restricted for passive mode of operation. It uses network layer markup mechanisms to avoid decapsulation of non-essential application data (stream segments). Also, a key difference is that we include the case of cooperative application processing in the service model where server side help may also be available. One can think EDIP is another index mechanism at the IP level beneath the application level for faster marker recognizing.

1.3 Layout of the Thesis

In Chapter2, we'll briefly introduce the framework of *active service distribution and Localization* (ASDL) model that provides application service between the end-user and the content-provider. We outline a framework that can support deployment of wide range of services with various specification and initiation dependencies. Then, in Chapter 3, we'll show the detail design of EDIP protocol and its working mechanism, which enables the high performance filter appliance. Finally, we'll examine the test bed of a fast filtering system based on EDIP in Chapter 4, and show the performance boost over conventional systems.

CHAPTER 2

ACTIVE SERVICE DISTRIBUTION AND LOCALIZATION MODEL

In this chapter, we propose the Active Service Distribution and Localization (ASDL) Model, which is an extension to CDN and CSN architectures. CDN defines a set of cache proxies distributed at the edge of Internet for faster data access. CSN extended the cache proxy concept to application proxy, which has the processing abilities for content adaptation rather than just receive-store-send function. Our ASDL model extends Ma's work, and makes co-operative computing available to content providers and service providers by distributing markers' programs and filters' programs between them. This ASDL model provides a systematic environment for EDIP enabled fast filtering and adapting.

2.1 Architecture and Components

The Active Service Distribution and Localization (ASDL) Model we propose identifies the following entities, as shown in Figure 1.

1. Service Management Server (SMS)
2. Adaptation Router (AR)
3. Content Provider (CP) and

4. End-User-agent (EU)

In this extended CSN infrastructure, the first two components play novel role. We provide a short description of each:

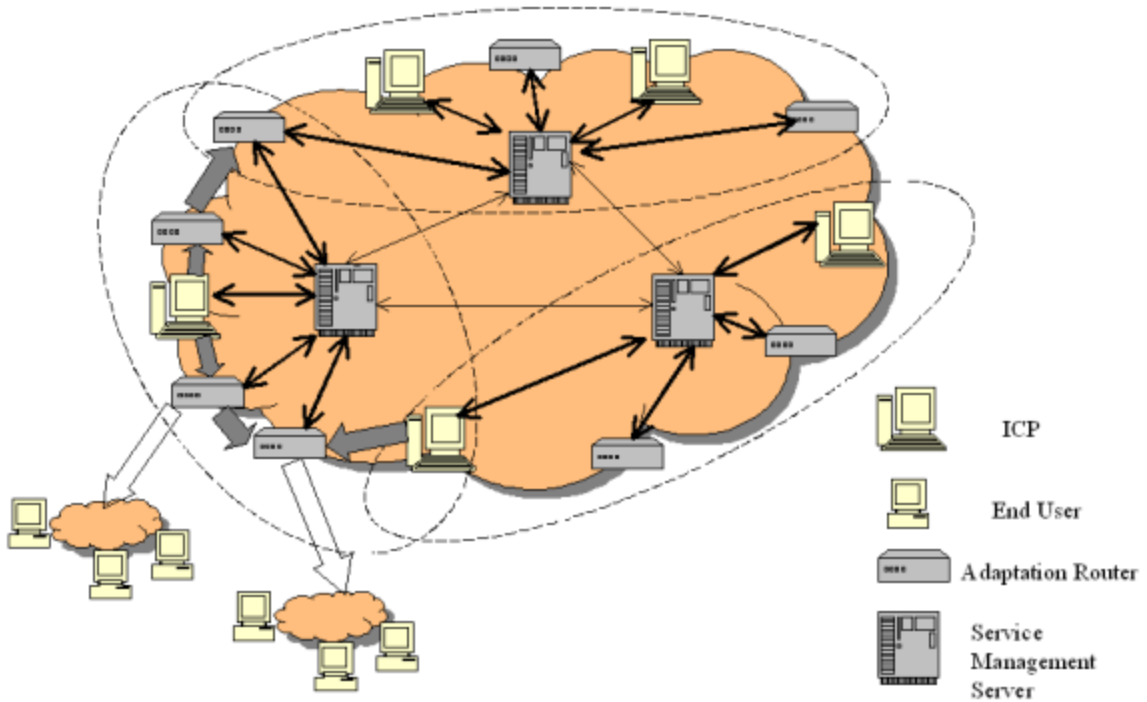


Fig 2.1: ASDL Architecture and Components

1. Service Management Server (SMS)

SMS serves as the principle service provider. They act as the mediation center among the end-users, adaptation router infrastructure providers and the content providers. The SMS owns the program modules called *switchlets* that are dynamically deployable to the ARs. These programs form the actual service. SMSs are responsible for the following tasks:

- (1) They maintain static and dynamic information about the service execution environment and the locations of the applications
- (2) They receive the service registration or cancellation requests from end-users, adaptation routers or content providers
- (3) Provide all authentication services
- (4) Aggregate the information about usage, availability and location of each deployed service, and then provide the information back to the deployment requester
- (5) Provide dynamic status visualization and monitoring, accounting and billing functionalities to value added service participating parties who use ASDL as an information exchange path
- (6) Each SMS is responsible for collecting information about its domain and periodically exchanges the information, such as registration and deployment status, with other cooperating SMSs. These exchanges can be triggered automatically if there is a change in the system.

2. Adaptation-Router (AR)

The adaptation routers are sparsely distributed special networked computing platforms, which, typically, will be deployed near the edge of the Internet. These can be setup as a service overlay and be owned by certain ISPs (or overlay ISPs). These ARs can

be rented as computational platform to process data streams either on behalf of content providers (CPs), the end users or even on behalf of the overlay ISP. Unlike the CSN's application proxy servers [6], these ARs can have special TCP/IP layers, which can enable them to fast intercept streams. The processing speed can be much higher than in application level, because (1) much less decapsulation, encapsulation work will be needed; (2) and simpler instructions in IP level will let us take advantage of RISC technology; (3) and some of data streams may indexed or marked by the corresponding ICP serverlet, for random access into the data stream. Chapter 3 describes the architecture of such a system.

3. Content-Provider (CP)

CP servers can be typical web servers. However, the protocol allows serverlets to be deployed at sender's location, if required by any service. For example a serverlet may pre-mark the outgoing data streams, when a particular service is active on the stream. The marker in a data stream can enable random access in adaptation routers (ARs), and therefore dramatically reduces the computation burden of ARs. We will show an active hyperlinking example in Chapter 3 and the saving and the cost will be shown and analyzed in our example.

4. End-User-Agent

EUs are the sinks/terminals of data streams. They may be the normal desktop/laptop

computers, or maybe handheld or wireless devices, or wearable computers. These terminals may have some kind of resource limitation, and therefore they need the resource or service provided by the ISP/AR. End-user agents generally maintain a resource-personalization specification, which can be polled by the SMS to determine the type and extensions of preprocessing requirements.

2.2 Information Components

Any service arrangement will require various types of information to be exchanged in various sequences among these parties.

The first form is the program elements (or the servelets and switchlets) those together create the service. A single service may require switchlets and serverlets to be deployed into multiple points.

These modules themselves also require additional parameters to run the service. The model identifies two types of such parameters. The *static* adaptation parameters are those can be received before the service begins. The dynamic adaptation parameters are those required with every request. We call this kind of parameters as *specifications*, and the party who send out the specifications as the specifier.

Example of static information includes *personalization cookie box* that contains a set of tablets containing the user, user-agent, and user-environment specific constraint information.

ASDL also allows *dynamic* custom index based random stream access. A servlet running on the content provider's site is a program that can be designed to help the service from the content source, such as source file indexing. An active application is a program that provides the service directly to the end-user, and it is designed to run on an adaptation router, which is normally controlled by some Internet service provider (ISP).

2.3 ASDL Contracting Model

The complexity of application service management grows because these information elements can come from variety of parties in various sequences based on the specific application service scenario. Before we introduce the ASDL protocol let us consider the issues: (1) who is going to supply the servlet running on the side of content provider and the active applications running on the side of service provider? (2) Who may be the service initiator? (3) Who are going to provide the parameter specifications?

Specfier	Destination of specification		
	EU	CP	AR(SP)
EU	No	Yes, by HTTP extensions or web forms	Yes, by HTTP extensions or web forms
CP	Yes, by XML or HTTP meta extensions	No	Yes, by serverlet
AR(SP)	Yes, by HTTP meta extensions	Yes, by active application program	No

Chart 2.1: Specification methods between different parties.

All the three parties (EU, CP and AR) can be initiators and parameter specifiers of the ASDL services. However, when the initiator and the specifier are the same party, there is no need for extra transmission. For example, if an end-user is requesting a bandwidth adaptation service, he or she can include the bandwidth information inside the initial request. However, transmission for dependent specifications between different parties is necessary. There are several ways to transmit the specifications: (1) by tightly coupled serverlet and active application programs (2) by XMLs or XML-like languages (3) by meta tags. The specifications between a CP and a SP can be expressed by method (1), because they share a couple of serverlet and active application programs, both of which derived from SMS. Information can be exchanged freely between the coupled programs.

The specification from content providers to the end-users can be expressed by the XMLs and HTTP meta extensions, while the specification from AR (SP) to the end-users can only be expressed by the HTTP meta extension. The CP and AR can make up web forms for the end-users convenience to provide the specification information. Chart 2.1 summarizes the discussion.

2.4 Classification of Active Services

From the service requesters' view, we may classify the services into two categories: (1) the single service request and (2) the group service request. A single service is requested by a single user and it will work solely for one user to meet its specific request. For example, a handheld device holder may request the adaptation router to translate all English web pages into German. This cannot be done at the handheld device, since it lacks memory, storage or processing speed to finish that task. In this case, the end-user may "buy" computation resource from the "net". The other type of service is group service, which is initiated either by the service provider or the content provider. For example, a service provider may have some agreement with the third party and advertise for them. The service provider then can analyze the web html files and put the ads at appropriate places. The group service can also be initiated by content providers. For example, a video source server may put special marks in the video stream and help the adaptation routers to downscale the video gracefully and meet the bandwidth requirement

for all different users. The service examples and the modes they belong to are listed below in chart 2.2.

Example of active services	Mode		
	Single Service	Group Services	
	EUI	SPI	CPI
Insertion of Ad Banners		*	
Multimedia adaptation for limited client bandwidth	*	*	*
Multi-language adaptation for different user preference	*	*	*
Active hyperlinking	*	*	
Active re-direction	*	*	
Virus Scanning	*		
Stream data adaptation and optimization	*	*	
Watermarking			*
Insertion of regional data	*	*	
Language translation	*		

Chart 2.2: A list of example services and their modes

2.5 ASDL SCENARIOS

2.5.1 EUI Model

In this scenario, the end-user initiates the service. Fig-2.2 illustrates the communication steps.

Setup Stage:

- (1) The EU sends service request to SMS.
- (2) SMS sends query to the participating ICP Source (ICPS) and AR to collect necessary configuration data. The query is with the identification of the SMS.
- (3) The ICPS and the AR response with digital signature for authentication and other necessary configuration information to SMS.
- (4) SMS then delivers the application modules to ICPS and AR, with corresponding security keys, which are required when installing the modules.
- (5) The ICPS and the AR send back the acknowledgements.
- (6) SMS sends the response back to EU with the certificates that EU may need when sending requests to AR and ICPS.

Data Transfer Stage:

- (A) EU sends request with certificates provided by SMS.
- (B) ICPS sends out data packages with EDIP headers.
- (C) AR processes the packages with EDIP headers, performs value-added in service, and sends result to EU with normal IP packages

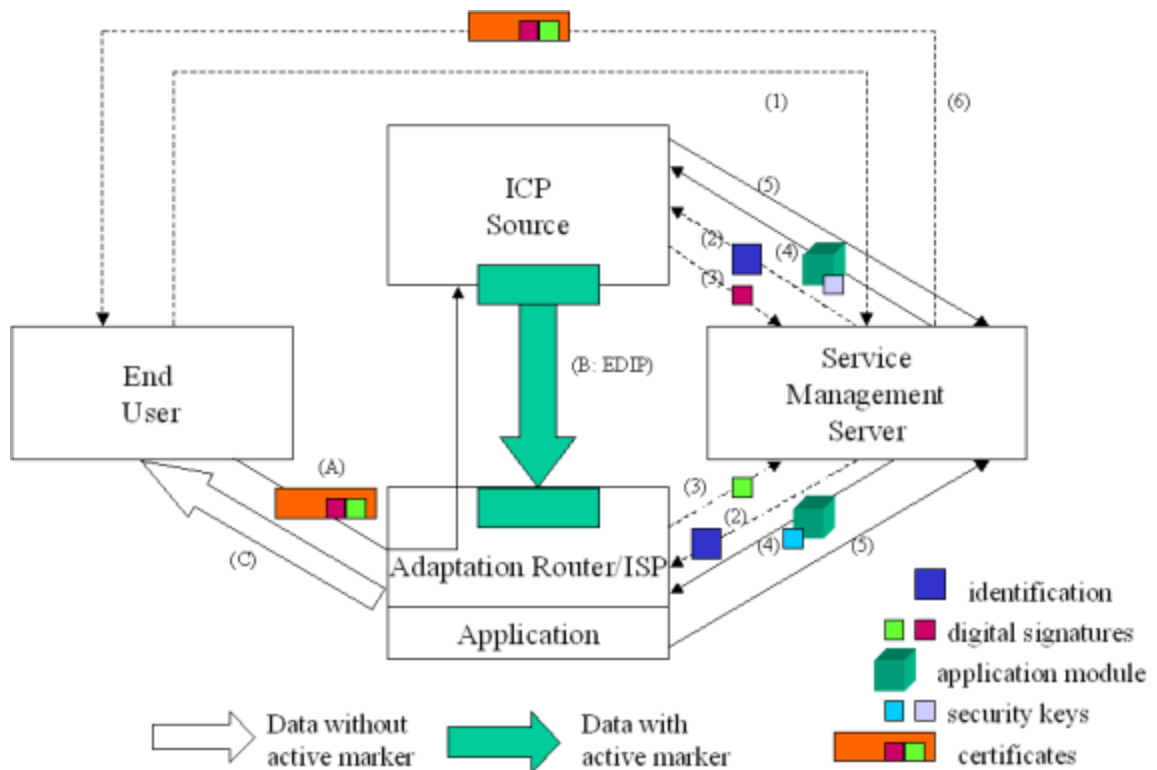


Fig 2.2: End-user initiates single service

2.5.2 CPI Model

In this scenario, the content-provider initiates the service. Fig-2.3 illustrates the communication steps.

Setup Stage:

- (1) The ICP Source (ICPS) sends service request to SMS.
- (2) SMS sends query to the participating ICPS and AR to collect necessary configuration

- data. The query is with the identification of the SMS
- (3) The ICPS and the AR response with digital signature for authentication and other necessary configuration information to SMS.
 - (4) SMS then delivers the application modules to ICPS and AR, with corresponding security keys, which are required when installing the modules.
 - (5) The ICPS and the AR send back the acknowledgements.
 - (6) SMS sends the response back to ICPS with certificates that ICPS may need when sending requests to AR.

Data Transfer Stage:

- (A) End-user (EU) sends the data request
- (B) ICPS sends out data packages with EDIP headers.
- (C) AR processes the packages with EDIP headers, performs value-added in service, and sends result to EU with normal IP packages

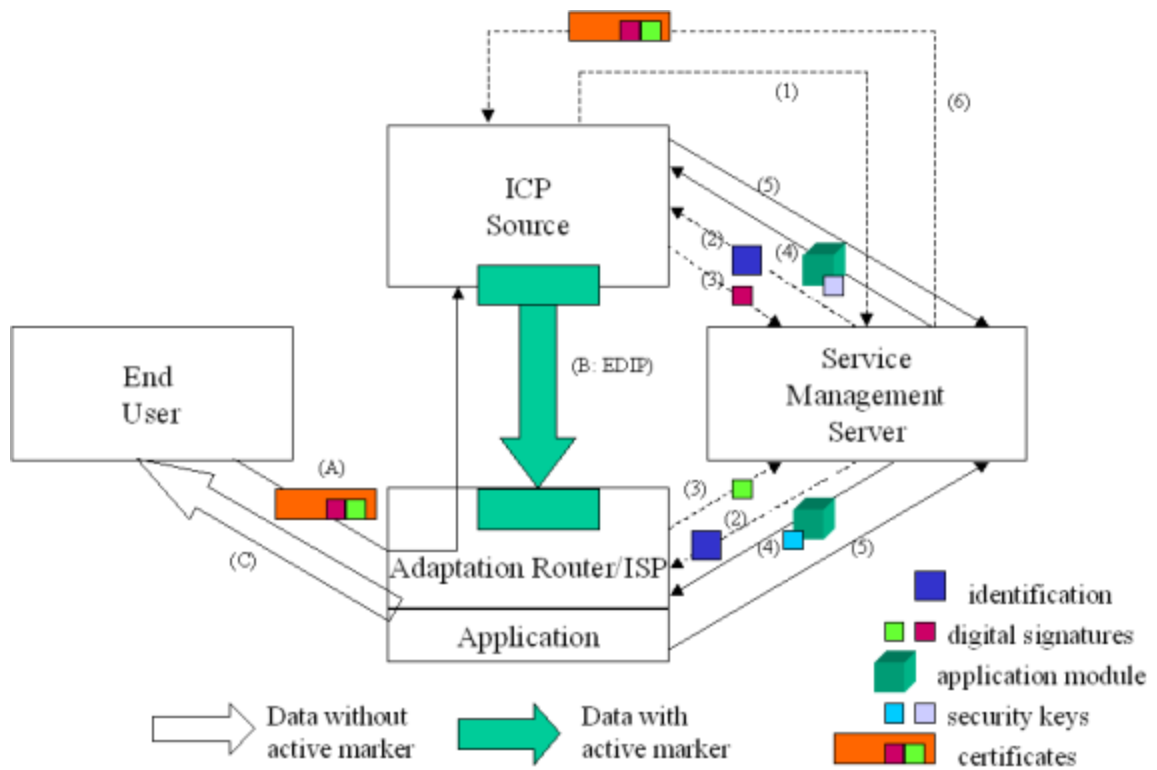


Fig 2.3: Content Provider initiates group service

2.5.3 SPI Model

In this scenario, the service provider itself initiates the service, and requests contracts from the content provider and adaptation routers. Fig-2.4 illustrates the communication steps.

Setup Stage:

(1) The Service Provider (SP) sends service request to SMS

(2) SMS sends query to the participating ICP Source (ICPS) and AR to collect necessary

configuration data. The query is with the identification of the SMS

(3) The ICPS and the AR response with digital signature for authentication and other necessary configuration information to SMS.

(4) SMS then delivers the application modules to ICPS and AR, with corresponding security keys, which are required when installing the modules.

(5) The ICPS and the AR send back the acknowledgements.

(7) SMS sends response back to ICPS with the certificates that ICPS may need when sending requests to AR.

Data Transfer Stage:

(A) EU sends the data request.

(B) ICPS sends out data packages with EDIP headers.

(D) AR processes the packages with EDIP headers, performs value-added in service, and sends result to EU with normal IP packages.

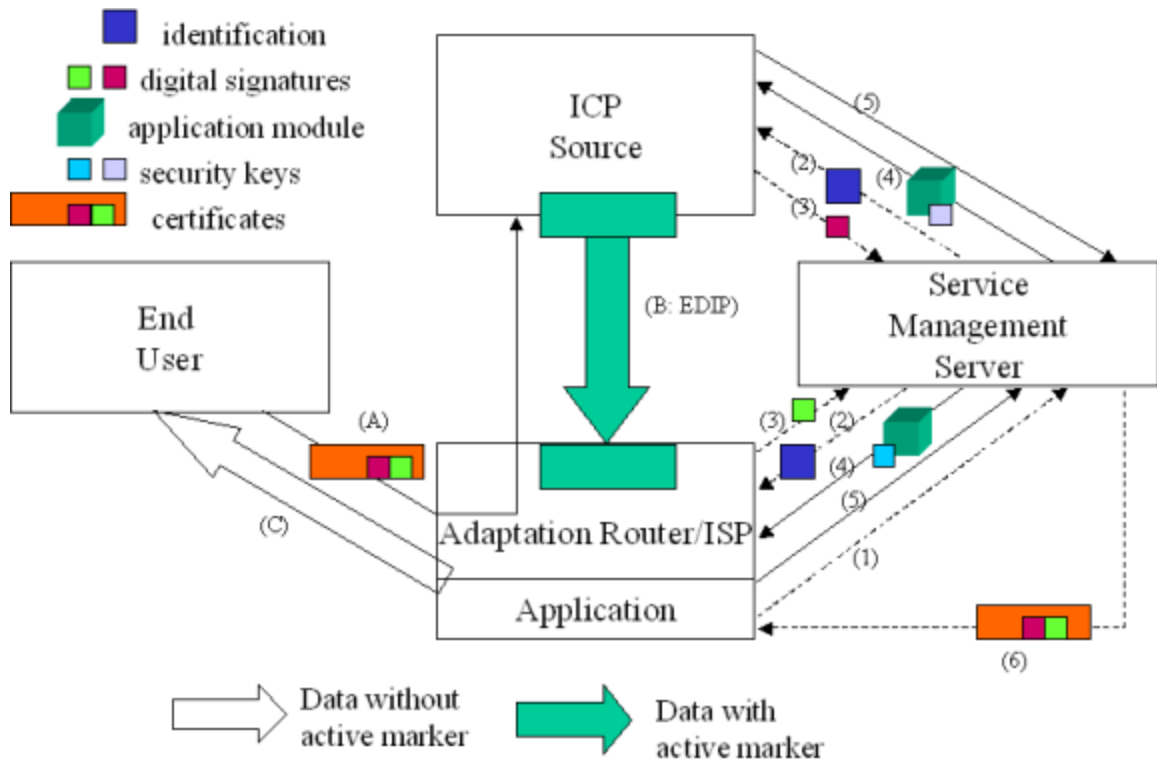


Fig 2.4: Service provider initiates group service

CHAPTER 3

EDIP PROTOCOL

In this chapter, we propose Embedded Data Indexing Protocol (EDIP). We will first introduce the concept of in route application service. Then we'll move to EDIP's indexing mechanism and its header format. We are also going to examine how the EDIP is encapsulated and decapsulated at content providers and service providers, respectively. A set of API is provided for easier user filter application developing. Finally, we'll go through a number of examples and see the result from some sample user plug-in applications.

3.1 In Route Application Service

First we explain the service model. In the service model a content stream from content provider's server (CP) flow to the end-user (EU). However it may also be processed in an ISP application processing (AP) server in between during transit. The end user initiates the content delivery by requesting content from the content provider via Internet. The Application Service Provider (ASP) modifies the content and adds value to the communication by application level intercept processing at strategically and/or topologically located AP servers. In special cases the CP and AP can be collocated in

application service provider's AP.

A special case of AP intervention is the passive filtering service where AP server only monitors the stream without changing it. A further special case is the stealth filters where servers or end-users are not aware of the intercept service (and thus also not helping). If the content provider is also willing to help we call it co-operative filtering (for non-co-operative filtering some extra fast string matching operations are needed at the AP server).

The AP server additionally can provide "content cache". The cache can connect at either 'pre' or 'post' AP stage. Conceptually, caching is just another piped service that AP can provide. AP server can be configured to provide multiple services piped on a specific request/response stream-- caching can be one of them. The piping sequence is soft configurable. Complex application service can be composed from simpler services by service piping. The connection between EU, CP, and AP servers are provided by point-to-point separate TCP/IP or UDP connections.

3.2 EDIP Indexing Mechanism

The operation of application processing is expedited by two techniques. The first is pre-marking the content stream and allowing fast access into to the stream. Second is the selective decapsulation re-encapsulation of only the pertinent data segments. Finally, we

also define a language to express and carry the marks between the parties involved.

The actual content intercept processing is performed by a program called the application *filter capsule*, and it runs on AP server. The application service provider generally also sends a *marking serverlets* to the CP server for marking of the content stream. Every Application Service Processing has a specified “*scope segment*” and a “*key segment*” in it. Generally a service is conditional. The data element which contains the condition or key is always intercepted and is decapsulated and delivered to the application capsule. The stream segment which is within the scope of an active key is intercepted and buffered. However, its decapsulation and delivery can be deferred based on the key evaluation result. If the evaluation is false, it is directly forwarded. Fig-1 shows the example service with EDIP header, and Fig-3 and Fig-4 are the schematics of the enhanced network layers that we have designed for the appliances machine.

3.3 EDIP Header Format

EDIP uses IPV6 extension header for content marking. It contains two parts: the General Field (GF) and Key Blocks (KB). The General Field (GF) identifies that it is an EDIP header, and contains general information in how to process the header. Each Key Block (KB) represents a keyword in this IP package, with positions of the keyword indexed by the offsets. Not every EDIP header has one or more KBs. Sometimes, an

EDIP header may only have a GF, representing that the current IP package belongs to an indexed stream, while there is no key word appearance in this package. The total number of KBs that an EDIP can have is only limited to the maximum size of an IP package. Fig 2.1 shows a possible EDIP header format.

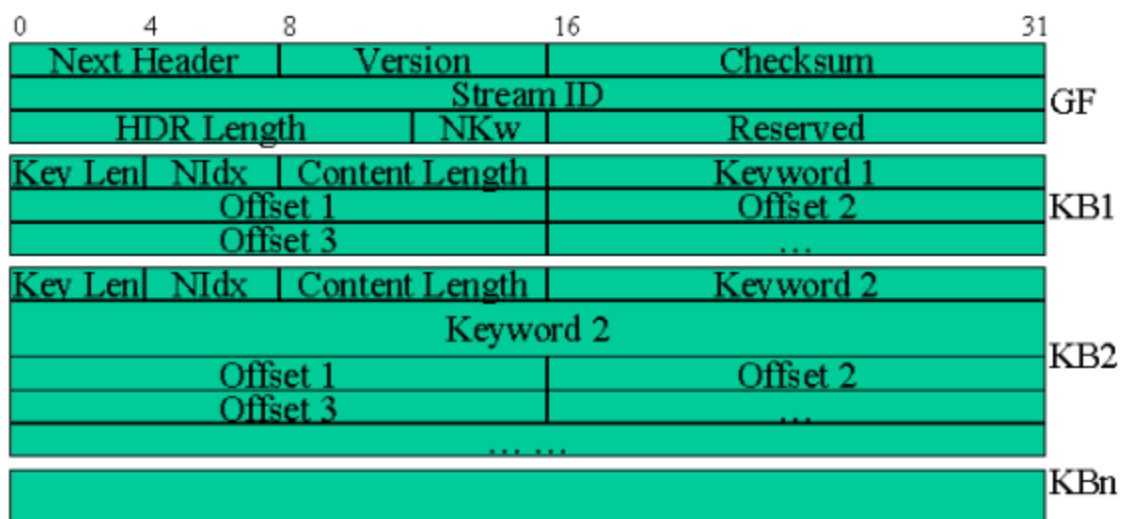


Fig-3.1: EDIP Header Format

Fields in EDIP are defined as below:

---General Fields:

Next Header: Next Header Types

Version: Version number of EDIP, the first bit indicating if it is encrypted or not

(1=encrypted, 0=not encrypted)

Checksum: Standard Checksum

HDR Length: The length of this EDIP Header, in words(4bytes).

Stream ID: Hash number of source port, destination port and sequence number from TCP header.

NKw: Number of Keywords included in this EDIP Header, 16 maximum

Reserved: Reserved for future use (for example, longer keyword length)

---Key Block Fields

Key Len: Keyword Length, in words, 16 maximum

Nidx: Number of Indexes for the keyword, 16 maximum

Content Length: The length, in bytes, of content immediate after the keyword, 256 bytes maximum

Offset: Location of the keyword in the ipv6 package

3.4 EDIP Encapsulation by Servelets

After capsulated by TCP/UDP, data stream can pass through multiple markers in the

source's servlet pipe. Each marker program is associated with exactly one keyword and it examines the passing stream to see if there is any keyword appearance inside. If there are one or more appearances, the marker generates a key block containing the offset information about where the keyword is in the stream. Later, these key blocks join the original data stream in the general field generator, where a GF, as well as the key blocks, will be added at the beginning of each package. The encapsulation process is shown below in Fig 3.2.

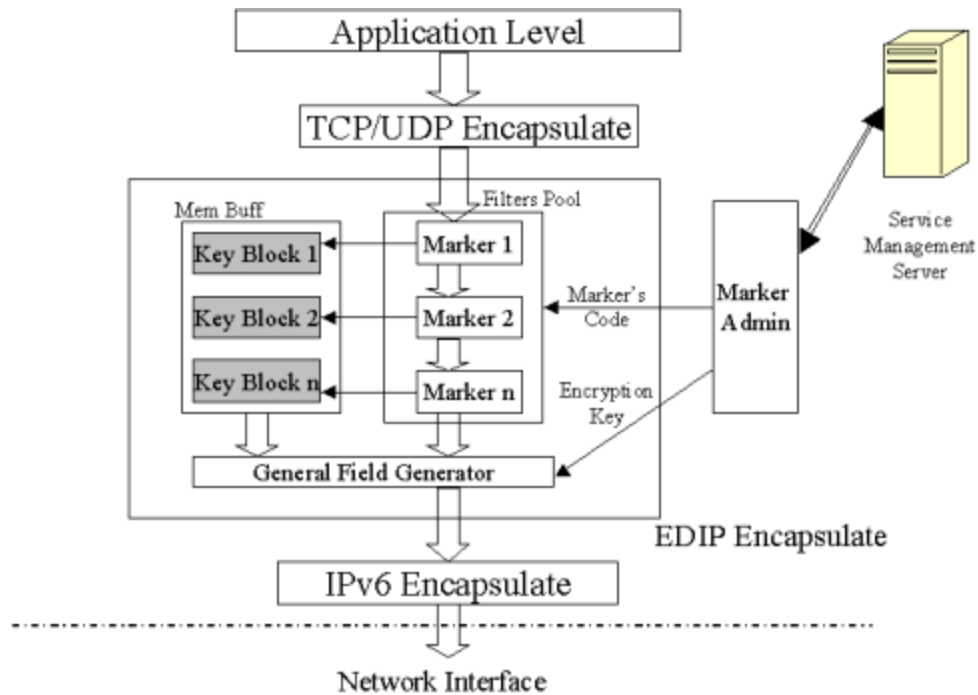


Fig-3.2: EDIP Encapsulation

The markers' codes are registered and distributed by SMS. Each CP's server running

the markers will have a marker admin (MA) to maintain the markers. After MA receives markers deployment request from SMS and pass the authentications, MA will check if there is available resource (such as available slots in markers' pool, the size limit of a marker, etc) to deploy the marker. To enhance the security, MA may provide an encryption key to the general field generator, who may encipher the GF, and only authorized value-added service providers can decipher it. A possible marker and GF generator's pseudo codes are presented below in pseudo-3.1 and pseudo-3.2.

Marker Code:

- Example: search a keyword --- "chess"
- Input: package-sized data from transport layer
- Output: a key block in memory buffer

```

Marker_n(m) //m: package-sized data
{
  Keyblock kb;
  kb.keyword="chess";
  kb.keylen=sizeof("chess");
  kb.contentlen=0 //specified by Coordinator
  I=0;
  while ((kb.offset[I]=nextpositionof(m, "chess"))>=0) I++;
  kb.Nidx=i+1;
  write kb to next blank key block lot in memory buffer;
}

```

Pseudo-3.1: Marker's pseudocode

A real marker program example at source side is attached at the end of this thesis.

The program used in the sample introduced in section 3.8.1.

General Field Generator:

- Input: package-sized data from transport layer, memory buffer for key blocks, encryption key from marker admin
- Output: package-size data with encrypted EDIP header

```
GField(m, kb[ ], e_key)
//m: package-sized data,
//kb[]: key blocks in memory buffer
//e_key: encryption key
{
    EDIP_H ediph;
    ediph.Nkw=num_of_kb_in(kb[ ]);
    ediph.hdrln=sizeof (kb[ ]);
    ediph.streamid=
        hashcode(m.s_port, m.d_port, m.s_ip, m.d_ip);
    ediph.version=0;
    ediph.nextheader=TCP/UDP;
    ediph.kbs=kb[ ];
    M_APPEND(ediph, m);
    fill_in_checksum(m);
    encrypt(m, e_key);
}
```

Pseudo-3.2: General Field Generator's Code

3.5 EDIP Decapsulation and Indexing/Scoping

EDIP decapsulation and value-added services are executed in the ISPs Adaptation Router (AR), which sit on the edge of Internet backbone. There are several tasks that an AR must do. (1) Differentiate the IP packages that need special processing from those normal IP packages. (2) Retrieve the offset information from the special-marked streams to the corresponding applications, which may use the information for value-added service. (3) Negotiate with SMS and maintain the service statistics. The main components include a stream controller, a keyword detector and a buffer controller.

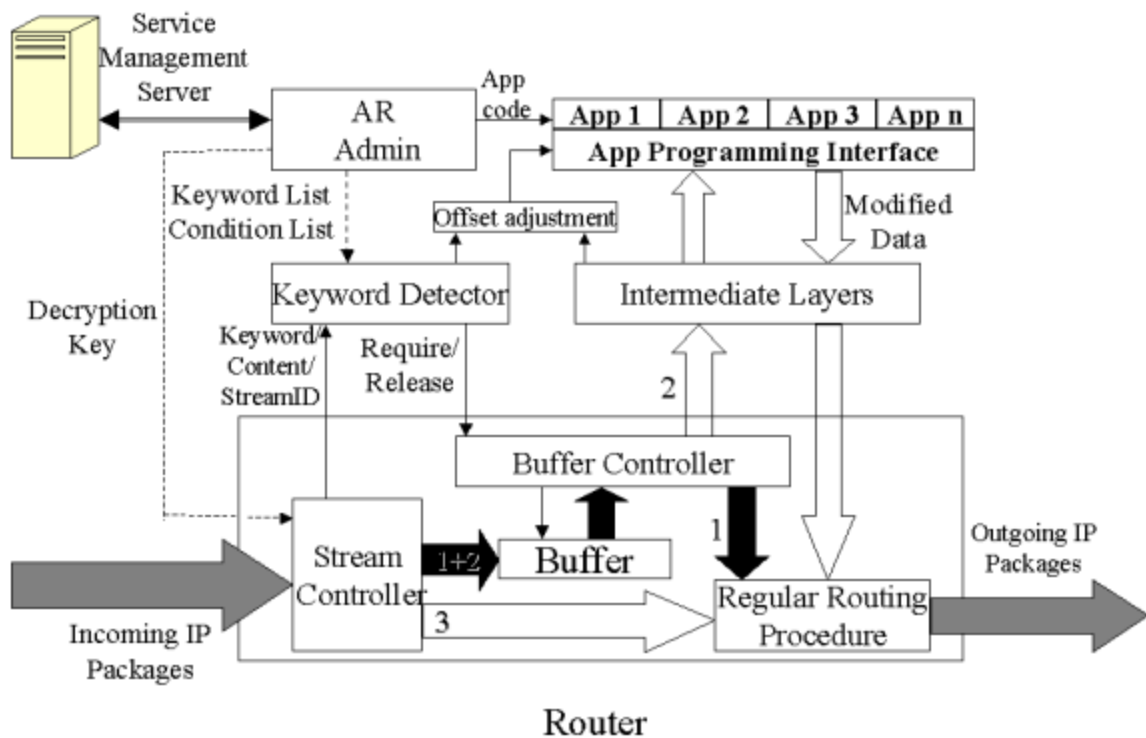


Fig-3.3 EDIP Selective Decapsulation System

Fig-3.3 shows a possible architecture of a typical selective decapsulation system running on a router. Its main components and functions are described as follows:

Stream Controller:

A stream controller's input is mixed IP packages, which may be IP packages with EDIP header, or just normal IP packages without EDIP header. A stream controller is supposed to forward those IP packages without EDIP in normal procedures, and store those with EDIP header into the Buffer for further actions. Further more, if the EDIP header contains any key blocks, the stream controller will decrypt it with corresponding decryption key from AR Admin, and send the keywords, contents and streamids to Keyword Detector. A possible pseudocode of a stream controller is shown below in Pseudo-3.3:

```

If (EDIP header doesn't exist)
    Send IP package to regular routing; //3

If (EDIP header exists) { //1+2
    If (EDIP header doesn't contain any key blocks)
        Put IP package in the buffer;
Else { //EDIP header contains some key blocks
    Decode the EDIP header with decryption key;
    Send keywords contents and streamid to Detector;
    Put IP package in the buffer;
}
}

```

Pseudo-3.3: Stream Controller's Code

Buffer Controller:

The buffer controller is supposed to maintain two lists --- a `required_streamid_list` and a `release_streamid_list`. Periodically, the buffer controller will check if there are any IP packages with the stream id listed in the two lists. Those in required list will be sent to application level and those in release list will be forwarded to their destinations. Every IP package in the buffer has a timestamp. If timestamp expires, the IP package will be released. A possible pseudocode of a buffer controller is shown below in Pseudo-3.4:

```

Require (streamid)
    if streamid is not listed in
    required_streamid_list[ ]
        add streamid into
        required_streamid_list[ ];

Release (streamid)
    if streamid is not listed in
    release_streamid_list[ ]
        add streamid into
        release_streamid_list[ ];

Buffer_Check()
    If ( IP package's streamid is listed
    in required_stream_id[ ])
        Send IP package to
        intermediate level; //2
    If ( IP package's stream id is
    listed in release_stream_id[ ])
        Send IP package to regular
        routing ; //1
    If ( IP package's timestamp expired)
        Send IP package to regular
        routing ; //1

```

Pseudo-3.4: Buffer Controller's code

Keyword Detector:

The keyword detector is supposed to check if the keywords sent by stream controller are in the keyword list maintained by AR Admin. If not, the stream id will be added into `release_streamid_list` in the buffer controller. If yes, the stream id can be added to the `required_streamid_list`. Sometimes, Detector can do a little more. For example, each keyword entry can have a condition on the corresponding content. If a package's content matches the corresponding condition, its stream id will be added in the `required_streamid_list` in the buffer controller. If not, release it. A possible pseudocode of a keyword detector is shown below in Pseudo-3.5:

```

DetectorQ
GetFromIP(I_Keyword, I_Content, I_Streamid);
If (I_Keyword is not in Keyword_list)
    release(I_Streamid);
elseif (I_Content matches corresponding
entry in condition_list)
    require(I_Streamid);
else release(I_Streamid)

```

Pseudo-3.5: Keyword Detector's Code

If we look back, we will find that the existence of EDIP header, in fact, plays the role of scoping facility, while the offsets in the EDIP header play the role of indexing facility.

By having these two facilities, the performance of applications running at adaptation

routers will be improved significantly, which will be shown in Chapter 4.

3.6 Application Processing

The application is armed with a set of special services APIs to take advantage of the marking processing.

These APIs can be viewed as two parts: (1) the administrative API subset, which is related to the start and stop of the service, and (2) the data manipulate API subset, which is related to editing the coming stream. An example of these two subsets of APIs is illustrated below in table-3.1 and table-3.2. The application program can use the administrative API subset to edit, bypass, drop, or insert bytes with a sequence stream of incoming data. The buffers are application buffers. Each of these operations is performed within the context of an incoming and outgoing TCP socket stream pair. Fig-5 shows an example of a stream-edit capsule and its edit operation on a stream. The stream offsets are algebraically calculated from key indexes supplied by EDIP. The data manipulate API subset can enable/disable the tracking of keys by activating/deactivating the marker/servelets and the intercept mechanism beneath. It can request for the next offset for a particular key. If the key test is successful (or unsuccessful), it can request (or release) delivery of the scope data. The AP capsules are also given a set of fast string search and protocol parsing routines (with potential hardware accelerators).

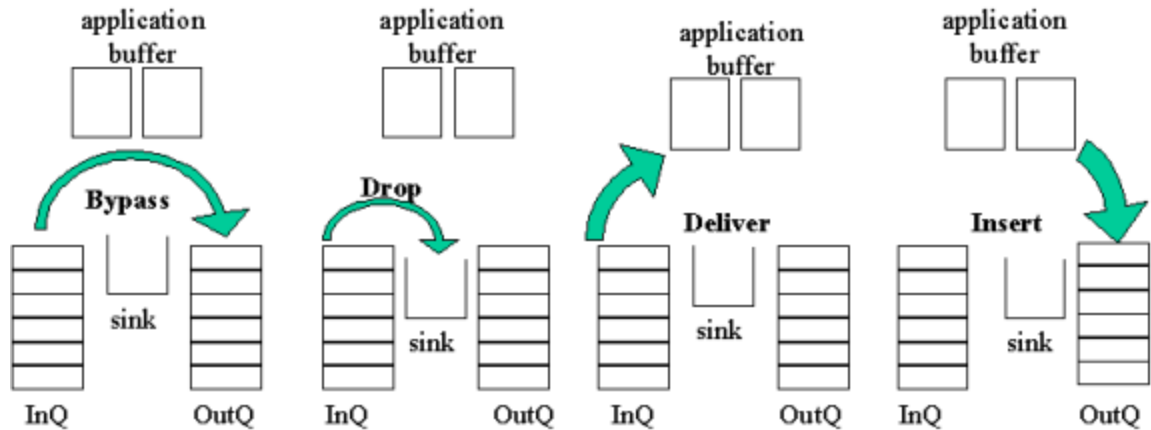


Fig-3.4: Data Manipulate API Operations

API	Comment
ActivateMarker(IP, M_ID)	Start the marker (serverlet) at source side
DeactivateMarker(IP, M_ID)	Stop the marker (serverlet) at source side
ActivateEditor(IP, E_ID, labellist, range)	Start the editor at router's side
DeactivateEditor(IP, E_ID, range)	Stop the editor at router's side
ActivateTrap(E_ID, labellist)	Set the trapper in OS

Table –3.1: Administrative API Subset

API	Comment
Associate (inQ, outQ)	Associate two streams
GetOffset (label)	Get the offset of the label in the stream
Bypass(sid, a, b)	Forward bytes from a to b
Drop(sid, c, d)	Drop bytes c to d (into trash sink)
Deliver(sid, e,f, &msgbuffer)	Deliver bytes from e to f with <i>newcontent</i>
Insert (sid, msgbuffer)	Insert the msgbuffer content to the stream.

Table –3.2 Data Manipulate API Subset

After the servlet and the filter have been deployed, a common procedure will be taken at the adaptation router's execution environment to conduct the service. Both the administrative API and the data manipulate API will be used in those procedures.

- (1) Activate marker (in servlet) at the source side. This step will activate the pattern detector, which will search some specific keywords or labels.
- (2) Activate system trap in the active router's execution OS, telling the OS when some keyword in the labellist comes, wake the service up.
- (3) Go to sleep
- (4) When waken up by the OS, request to deliver the stream within the specified range to the application
- (5) The application will use data manipulate APIs, such as `getoffset()`, `bypass()`, `insert()`,

drop() and etc to modify the data stream if needed. An example of a content processing using stream edit API is show in Fig-3.5.

(6) Go to step (3) until the editor is deactivated.

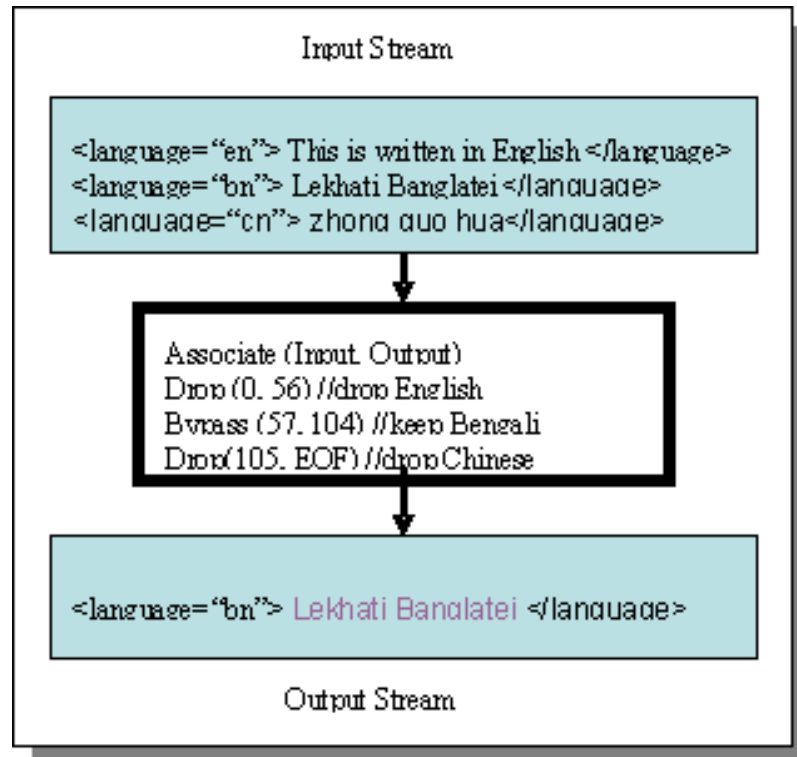


Fig 3.5 Example of Content Processing with stream-edit API

For example, if we want to process the stream shown in Fig-3.5 for language translation, one possible procedure will be taken as shown in Pseudo-3.6. The labellist used in the pseudo code is shown in table-3.3. The offset information in table-3.3 is retrieved after 'GetOffset' API is executed in pseudo-3.6.

M_ID	Keyword	Offset
1001	<language="en">	0
1002	<language="bn">	57
1003	<language="cn">	105
1004	</language>	45, 93, 133

Table-3.3: The labellist used in the example shown in Fig 3.6

```

ActivateEditor(localhost,
               39342, labellist, Full_Range);
ActivateMarker(SourceIP, 1001);
ActivateMarker(SourceIP, 1002);
ActivateMarker(SourceIP, 1003);
ActivateMarker(SourceIP, 1004);
ActivateTrap(E_ID, labellist);

LOOP:
Sleep until waken up;
GetOffset (labellist);
Associate(Input, Output);
Drop(0,56);
Bypass(57, 104);
Drop (105, EOF);
If deact is true, exit LOOP;
END LOOP;

DeactivateTrap(E_ID, labellist);
DeactivateMarker(SourceIP, labellist);
DeactivateEditor(localhost);

```

Pseudo-3.6: A pseudocode for processing examples shown in Fig-3.5

To get friendlier programming interface, these APIs may be wrapped for easier use. At the end of the thesis, we attached a simple filter application, which adds links to some specific keywords appeared in passing-by HTML streams, to show how the APIs and wrap-up work. The result of the filtering is going to be shown in Section 3.8.1.

3.7 An example illustration

The proposed mechanism accelerates the application level intercept process. The advantage is derived essentially by three principal sources: (1)Only the byte segments carrying ‘keys’ are unconditionally decapsulated. (2)The byte segments carrying ‘scope’ are conditionally decapsulated only when the key conditions are true. (3)Rest of the bytes are never decapsulated.

There is also another source of run-time performance boost. Stream is marked by the servlet processes running at the content source. In cases, it is sometime possible to mark with direct content knowledge by the content generator without any string search. Otherwise, the marking can still be performed by sting search/ or parsing of the original content as preprocessing. It still therefore can drastically reduce the run time cost. To compare—current filters have to perform run-time full search and/or full parsing. The scheme however has cost. It is the extra data that will be needed by the EDIP markers. The actual saving therefore is the function of *key density*, and the *key success probability* in the stream. Though, apparently it may seem that high key density can offset the performance gain, but in practice the EDIP key density can always be controlled by using a gross key in EDIP and then using application level processing to find the real keys. This is benefit of application level soft key definition ability. In practice, only a small part of data stream is generally modified. Consequently, the expensive part is way too

inconsequential compared to the saving made by bypassing the costly decapsulation/reencapsulation of the rest.

Here we introduce an active hyperlinking example, which will add a corresponding hyper link when it meets some specific word. It involves three parties: (1) an end user, who is requesting several files from content provider via Internet; (2) a content provider, say, CNN.COM, which provides the original data and runs the serverlets on one of its servers generating the EDIP header; (3) a value-added service provider, normally known as an ISP, say, AOL.COM, which owns the ARs. The scenario is shown below in Fig 3.6.

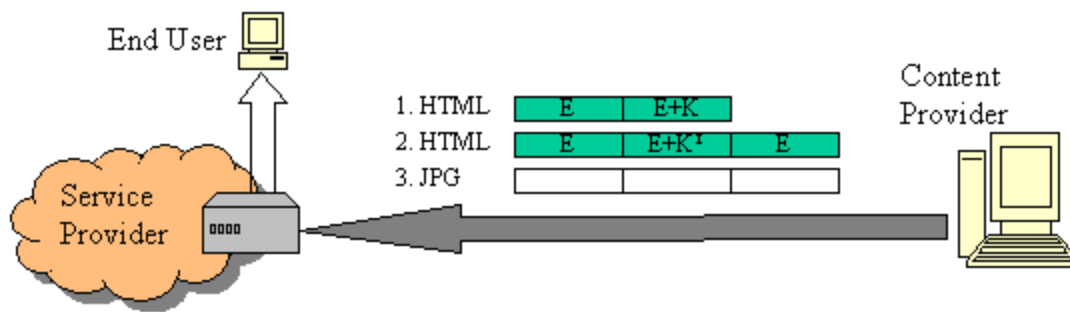


Fig 3.6 An example service

Assumptions:

- A user of AOL is requesting two HTML files and one JPG file from CNN.COM.
- AOL's online mall is selling motherboards, and they want to put hyperlinks on where the word "motherboard" appears in HTML file.

- AOL has an agreement with CNN.COM --- CNN will put a marker where the word “motherboard” appears.
- The first HTML file is divided into 2 IP packages, the second HTML file is divided into 3 IP packages, the JPG file is divided into 3 IP packages
- The first HTML file has an EDIP header with keyword “shirt”, requested by other entity. AOL is supposed to ignore it.
- The second HTML file has an EDIP header with keyword “motherboard”, which is the keyword target, appearing in its second IP package
- The JPG file does not have any EDIP header

Mission:

The AOL adaptation router modifies all HTML files with “motherboard” by adding a link to its online mall. All other files are not supposed to change.

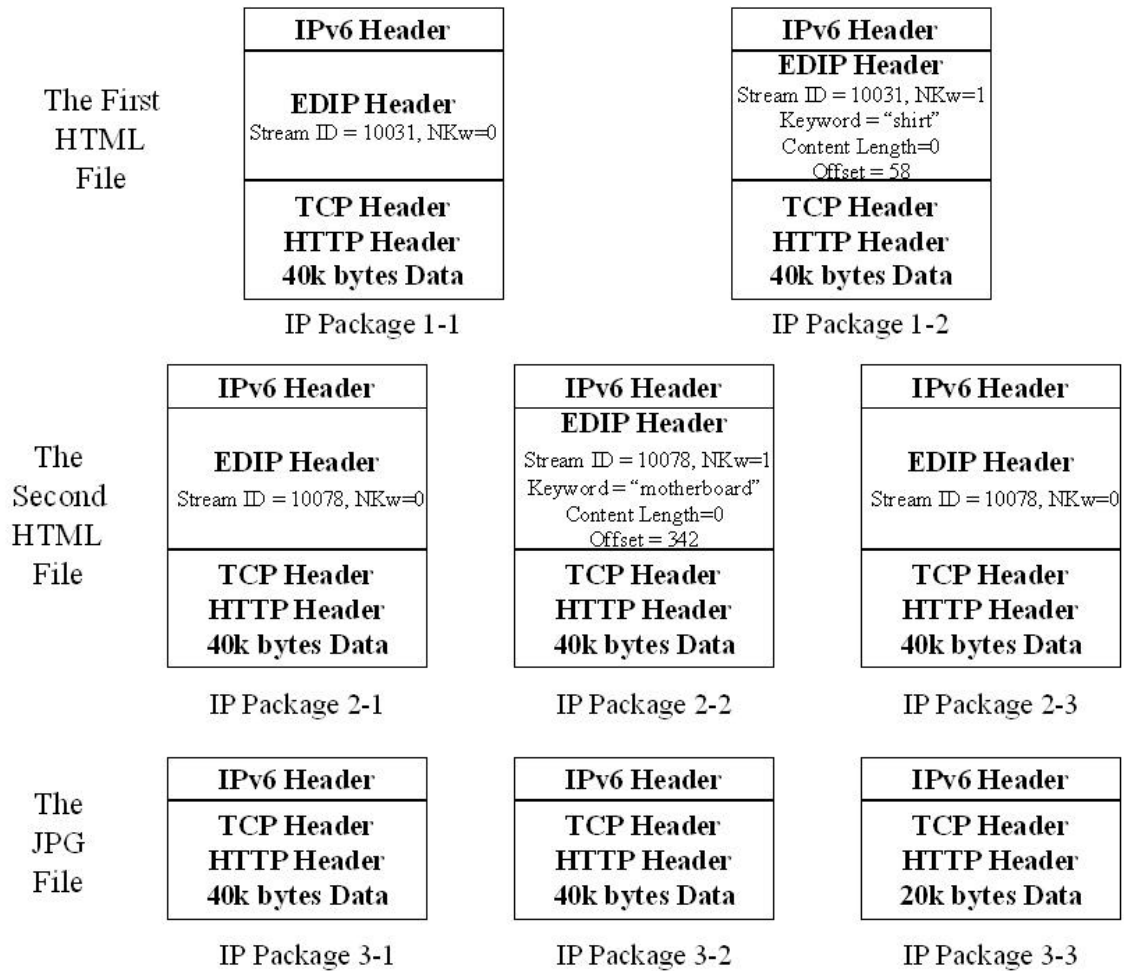


Fig 3.7: IP packages encoded with EDIP headers

Fig 3.7 shows the result after the 3 files have been processed by CP's serverlets. Each of the two HTML files has a keyword, ("shirt" for the first one, "motherboard" for the second one.) and each of them is carrying an EDIP header. The JPEG file does not have any keyword, and therefore no EDIP header is added. Both keyword "shirt" and "motherboard" appear in the second IP package in their own HTML files, and each of

these IP packages is carrying a general field and a key block. All the other IP packages from those two HTML files are only carrying general fields, which indicate they belong to a stream with keywords, but those keywords do not shown in the current package.

Chart 3.1 shows the EDI-Filtering (EDIF) process actions in AR for each IP package in our example. IP packages with EDIP headers (1-1, 1-2, 2-1, 2-2, 2-3) will be sent to the keyword detector, but only those match the requirement from applications will be decapsulated and sent to upper level for further processing. All the other IP packages will be forwarded as normal packages.

Chart 3.2 shows the processing actions in AR without EDIP header, i.e., the Full Search Filtering (FSF). In this schema, each IP packages coming into the AR will be decapsulated, tested, encapsulated, and forwarded, which is computation resource consuming compared to EDIF model. The gray blocks in Fig 7 and Fig 8 show the net saving of EDIF over FSF in our example. The yellow blocks in Fig 7 show the possible net cost, which is the task to detect if key words are matched. However, in FSF model, the task has also to be done, but it is often done in the application level. Based on that, we can almost neglect the cost of detector's in our future quantity analysis in performance.

Package	Actions and Destinations (EDIF mode)				
	Router Level	Intermediate	App Level	Intermediate	Router Level
1-1 [E]	To buffer				Forward
1-2 [E+K]	To detector To buffer				Forward
2-1 [E]	To buffer	Decapsulate	Modify Add a link	Encapsulate	Send
2-2 [E+K ^T]	To buffer	Decapsulate		Encapsulate	Send
2-3 [E]	To buffer	Decapsulate		Encapsulate	Send
3-1	To forwarder				Forward
3-2	To forwarder				Forward
3-3	To forwarder				Forward

Chart 3.1 Actions in EDI-Filtering (EDIF)

Package	Actions and Destinations (FSF mode)				
	Router Level	Intermediate	App Level	Intermediate	Router Level
1-1	Send up to Inter Layers	Decapsulate	Search Not Match	Encapsulate	Forward
1-2	Send up to Inter Layers	Decapsulate	Search Not Match	Encapsulate	Forward
2-1	Send up to Inter Layers	Decapsulate	Search	Encapsulate	Send
2-2	Send up to Inter Layers	Decapsulate	Match Modify	Encapsulate	Send
2-3	Send up to Inter Layers	Decapsulate	Add a link	Encapsulate	Send
3-1	Send up to Inter Layers	Decapsulate	Search Not Match	Encapsulate	Forward
3-2	Send up to Inter Layers	Decapsulate	Search Not Match	Encapsulate	Forward
3-3	Send up to Inter Layers	Decapsulate	Search Not Match	Encapsulate	Forward

Chart 3.2: Actions in Full Search Filtering (FSF) Mode

Actions	EDIF	FSF
Decapsulate	3 pkgs (2-1, 2-2, 2-3)	8 pkgs (all)
Encapsulate	3 pkgs (2-1, 2-2, 2-3)	8 pkgs (all)
Search	2 index searches (1-2, 2-2)	8 sequential searches (all)

Chart 3.3 Different Actions between EDIF and FSF mode

Chart 3.3 shows the difference in how much the computational resource is consumed between EDIF model and FSF model for our example.

3.8 Some Sample Plug-in Applications

In this section, we'll show three sample applications, which are implemented in our experimental adaptation routers, using EDIP indexing scheme for fast content interception and adaptation.

3.8.1 Active Hyperlinking

Active hyperlinking is adding hyperlinks to some specific patterns appear in certain web pages to draw the attention or provide more information to potential interested readers. The web pages are adapted neither at the content providers servers nor at the end users' computers, but at the adaptation server in between. Using this scenario, it will be

easier to localize or personalize the web pages the end users are going to see.

Fig 3.8 and Fig 3.9 illustrate the example. Note that before adaptation (Fig 3.8), the authors' names are plain text in "Publications" section. The plug-in program on top of the adaptation router then takes the authors' names as keywords, and changes the web pages by adding their corresponding email addresses wherever the keywords appear. Now, web page readers just need to click the links to send emails to the authors. (Fig 3.9)

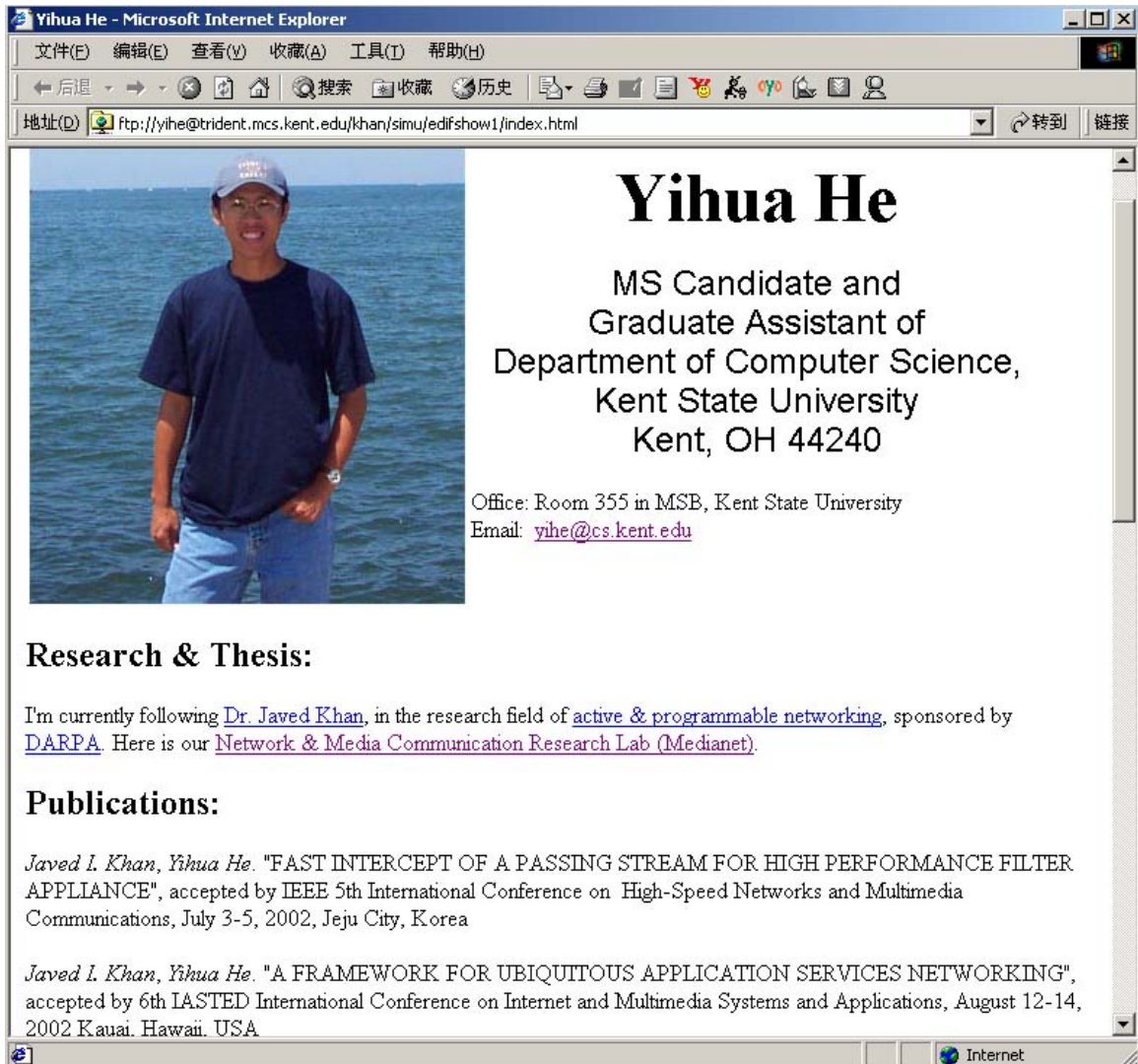



Fig 3.8 A Web Page before adaptation

Yihua He - Microsoft Internet Explorer

文件(E) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

← 后退 → 搜索 收藏 历史

地址(D) ftp://yihe@trident.mcs.kent.edu/khan/simu/sinkshow/index.html 转到 链接



Yihua He

MS Candidate and
Graduate Assistant of
Department of Computer Science,
Kent State University
Kent, OH 44240

Office: Room 355 in MSB, Kent State University
Email: yihe@cs.kent.edu

Research & Thesis:

I'm currently following [Dr. Javed Khan](#), in the research field of [active & programmable networking](#), sponsored by [DARPA](#). Here is our [Network & Media Communication Research Lab \(Medianet\)](#).

Publications:

[Javed I. Khan, Yihua He](#). "FAST INTERCEPT OF A PASSING STREAM FOR HIGH PERFORMANCE FILTER APPLIANCE", accepted by IEEE 5th International Conference on High-Speed Networks and Multimedia Communications, July 3-5, 2002, Jeju City, Korea

[Javed I. Khan, Yihua He](#). "A FRAMEWORK FOR UBIQUITOUS APPLICATION SERVICES NETWORKING", accepted by 6th IASTED International Conference on Internet and Multimedia Systems and Applications, August 12-14, 2002 Kauai, Hawaii, USA

完成 Internet

Fig 3.9: A Web Page After Adaptation

3.8.2 Advertisement Filtering

Today most content providers are bringing commercials to the web pages. Some people may not want to see those commercials, or they want to see more local news. By submitting the service to adaptation routers, the end users may get ad-free web pages, or, if ISPs wish, local advertisement can be inserted.

Fig 3.10 shows a Yahoo page before adaptation. The shaded areas are commercials marked by Yahoo. Fig 3.11 shows the web page after filtered. The indexing information in EDIP headers directs the filter which part should be deleted from the stream in order to get an ad-free page.

The same scenario can be used in parental control.

The screenshot shows the Yahoo! homepage in Microsoft Internet Explorer. The browser window title is "Yahoo! - Microsoft Internet Explorer". The address bar shows the URL: ftp://yihe@trident.mcs.kent.edu/khan/simu/edifshow2/Yahoo!.htm. The page features the Yahoo! logo and several navigation icons: Finance, Messenger, Check Email, What's New, Personalize, and Help. A banner for "Dial-up Internet Access First Month Free" is visible, along with a promotion for "FIFAworldcup.com" as the "OFFICIAL SITE OF THE 2002 FIFA WORLD CUP". A search bar is present with a "Search" button and a link to "advanced search". Below the search bar, there are links for "Yahoo! Sports" (U.S. Open, World Cup, NHL Playoffs, NBA Finals, Major League Baseball) and a "Shop" section with various categories like Auctions, Autos, Classifieds, Real Estate, Shopping, Travel, Yellow Pgs, Maps, Media, Finance, News, Sports, Weather, Connect, Careers, Chat, GeoCities, Greetings, Groups/Clubs, Mail, Members, Messenger, Mobile, Personals, People Search, Photos, Personal, Addr Book, Briefcase, Calendar, My Yahoo!, PayDirect, Fun Games, Horoscopes, Kids, Movies, Music, TV, and more... A "Yahoo! Shopping" section offers a search interface with "Shop by" options (Dept: Gifts, Store: Sony) and a "Search" field. It also features a "NEW Sony WEGA TV! Only \$22/month. Plus FREE In-Home Delivery." and a "Barnes & Noble - Free Shipping on 2 Items or More" promotion. The "In the News" section lists several headlines, including "U.S. military plane crashes near Gardez", "New al-Qaida warnings cited overseas", "Senate rejects repeal of estate tax", "Firefighters battle huge Colo. blaze", "Fashion designer Bill Blass dies", "Astronauts set space endurance record", and "Lakers sweep Nets for threepat". The "Marketplace" section promotes "DVD Players", "Polaroid Digital Camera" (only \$79.99), and "Polaroid PhotoMAX" (includes 3MB memory card, case, cables, AC adapter and more. You save 68%). The "Broadcast Events" section is also visible at the bottom. The page footer includes "Arts & Humanities", "Business & Economy", "Computers & Internet", "Education", "News & Media", "Recreation & Sports", and "Reference".

Fig 3.10: Before Adaptation: A Yahoo web page with commercials

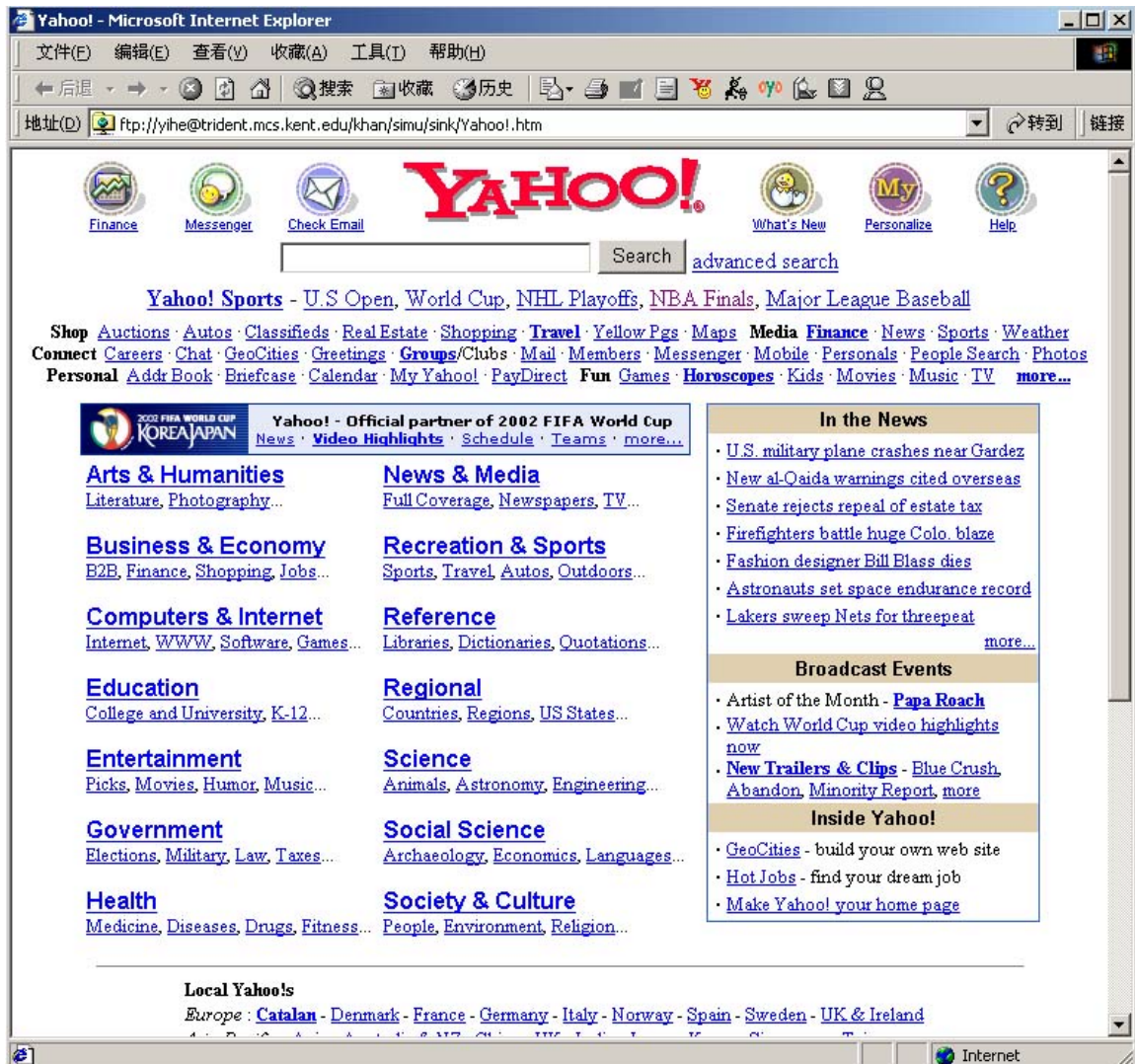


Fig 3.11 After Adaptation: An ad-free Yahoo page

3.8.3 Screen Size Adjustment and Re-laying



Fig 3.12: A web page from Yahoo before adjusting the size and relaying

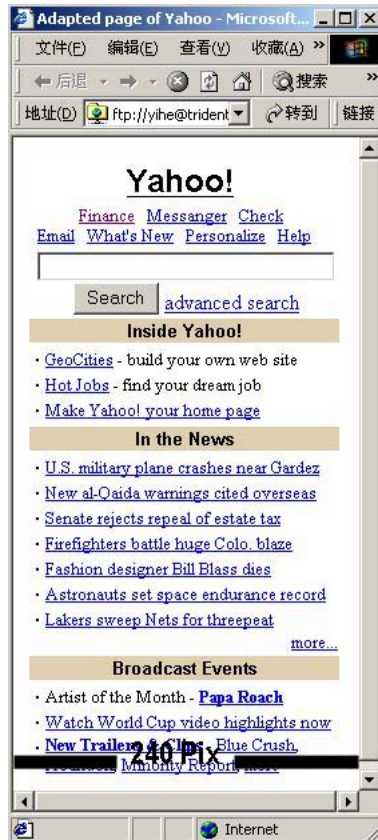


Fig 3.13a: After size adjustment1

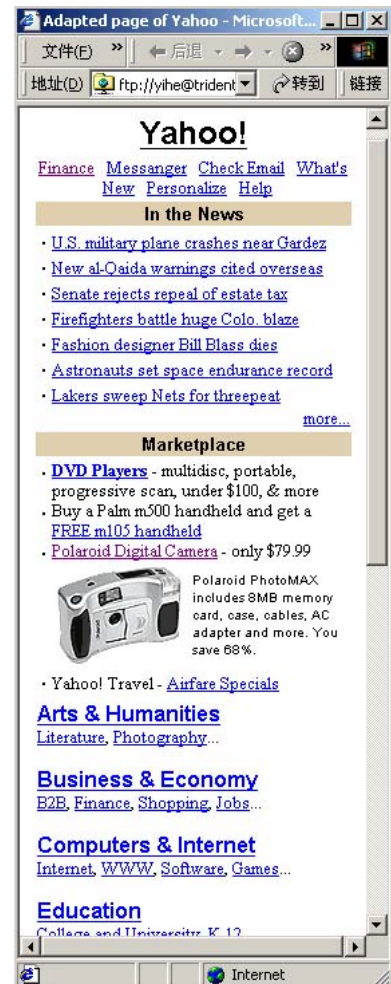


Fig 3.13b: After size adjustment2

Wireless devices are getting more and more popular. PDAs, pocket PCs, and even cell phones now can be used to surf the Internet. With limitation in dimensions, the viewable screen size of a hand-held device cannot be the same as that of a desktop or laptop computer. A mobile user will have difficulties in viewing a normal 800*600 page on a 240*320 screen size PDA. Nowadays, content providers often keep special “small screen” versions for mobile users. But with rapid growing diversity of hand-held devices,

this solution cannot meet all users' needs. Fig 3.12 shows a Yahoo page before screen resizing. The shaded areas are marked by special markers, which are keywords the adaptation router is looking for. Fig 3.13 shows the page after screen resizing. Note that the layout could be changed according to user's preference. For example, in Fig 3.14, the appearance and the sequence of appearance of memory blocks are different from those in Fig 3.13, although they are originating from a single source file and the adaptation plug-in doesn't change. (The user preference submitted to the adaptation router changes.)

CHAPTER 4

THE PERFORMANCE

An experimental test bed is designed to verify the functions and exam the performance of the adaptation router and the EDIP protocol we proposed above. We'll first introduce the hardware and software environments, on which we build the test bed. Then we'll investigate several performance benchmarks and compare them with those of conventional adaptation mode. Finally, we examine EDIP's extra space cost and the potential performance boost by using RISC technology and dedicated chips.

4.1 The Environments

The source code of the test bed was written in standard C language. The test bed has been successfully functioning in some of the major UNIX environments in the Department of Computer Science and Network & Media Communication Research Lab in Kent State University. The tested systems include:

- (1) TRIDENT, HP-UX B.11.00, Hewlett-Packard 9000/785
- (2) AEGIS, HP-UX B.11.11, Hewlett-Packard 9000/770
- (3) FORRESTAL, Redhat 7.1, AMD Athlon 800MHz

- (4) IOWA, Redhat 7.1, AMD Athlon 800MHz
- (5) AWAGATEWAY, Redhat 7.2, Intel Pentium 166MHz
- (6) DAVELINUX, Redhat 7.2, Intel Pentium 166MHz.

When we exam the performance of the adaptation router, we choose Linux Redhat 7.2 (Kernel 2.4.7-10) which runs on an Intel Pentium 166Mhz system. The reason we choose such a system to exam the performance of our adaptation router is described in the following sections.

4.1.1 Software Environment

The software environment we choose in the performance test for the adaptation router is Linux Redhat 7.2 (Kernel 2.4.7-10). At the time the test bed was built, version 7.2 was the latest distribution of Redhat Linux (now Redhat 7.3 has just landed), which included the 2.4.7-10 kernel. It supports up to 64GB of RAM, far more than the 4GB limit in the 2.2 kernel series. While the 2.2.x kernel can't take full advantage of servers with more than four CPUs, the 2.4 series is much more scalable, with SMP (symmetric multiprocessor) support for machines with as many as eight CPUs. From personal production to basic web serving, Red Hat Linux contains everything needed for a stable and secure working environment. With its powerfulness and versatility, we believe building our test bed on such a system has general and comparable meaning.

4.1.2 Hardware Environment

The hardware we use in the test bed is generally IBM-compatible PCs including Intel Pentium CPU and AMD Athlon CPU. The computer running adaptation router in the performance test has a single Intel Pentium 166MHz CPU and 64Meg SDRAM, with a conventional 3COM 10/100 network adapter. Other computers playing the content providers and the end-user agents include one Intel Pentium 166 MHz, one Intel Pentium IV 1.6 GHz and two AMD Athlon 800 MHz machines. The reason we choose a relatively slow system as our adaptation router is that, the `times()` function, which returns the processor time used when a process calls it, only has a resolution of 1/100th second. In this case, a slower system will produce less relative error. Further more, a fast system may produce too much data in too little time for a conventional 10/100M network adapter. The adaptation router will spend considerable resource on congestion control and make the execution time unpredictable. Using a slower system can avoid such unpredictable factors.

4.2 Test Application and Sample Used

The application used in this performance test is active hyperlinking. Two sample files are used. The first one is an HTML file, in which there are two keywords in our adaptation router's service range. Two hyperlinks will be added into the HTML file when it passes the adaptation router. This HTML file represents streams that need to be

serviced. The second is a JPG file, which represents streams that need not to be serviced. It will not be changed when it passes the adaptation router. Both files are trimmed to 50.0k bytes in size for easier performance calculation. Keeping same total number in amount, these two files will be sent repeatedly to adaptation routers in various ratios, simulating different service densities.

For comparison purpose, we also built two other models besides the Embedded Data Indexing Filtering (EDIF) mode. They are Full Search Filtering (FSF) service model and Normal Router (NR) mode, which is without any adaptation service. To make the result more comparable, these three models are built using the same programming strategy ---in fact, they share most codes in common parts, such as encapsulation, decapsulation, regular routing algorithms and so on.

4.3 Performance Test

4.3.1 CPU usages for EDIF service:

In this section we provide the performance of the EDIF filtering. We found that the CPU usages is closely related to the amount of data that in service range. We define “service density” as the percentage of data volume that needs adaptive service (i.e. in service range). In this experiment, we send total 5M bytes data through the adaptation router with variety of service densities. We plotted x axle as the service density ranging

from 0% (idle) to 100% (full). Fig 4.1 plots the absolute value of CPU time cost by EDIF schemes for major components in the adaptation router, and Fig 4.2 plots each component's relative CPU cost percentage, which is the ratio of CPU time used for this component to the total CPU time used by the adaptation router.

We can see from those graphs that with the increase of service density, the CPU time used by each major component is increased. The encap/decap time increases because the more IP packages in service range, the more IP packages need to be decapsulated and encapsulated for searching and modifying. The similar reason applies to the explanation why user application CPU time increases while service density increases --- this is due to more packages are in service range, the more packages need to be searched and modified. The routing time does not start from 0 when the service density is 0%. This is because the adaptive router has to spend CPU time in regular routing. When service density increases, more packages have to be routed to buffers and queues and delivered to user application for adaptation purpose. Those packages take more routing time than the packages that just need regular routing to their network destinations. That explains why the routing time also increases when the service density increases.

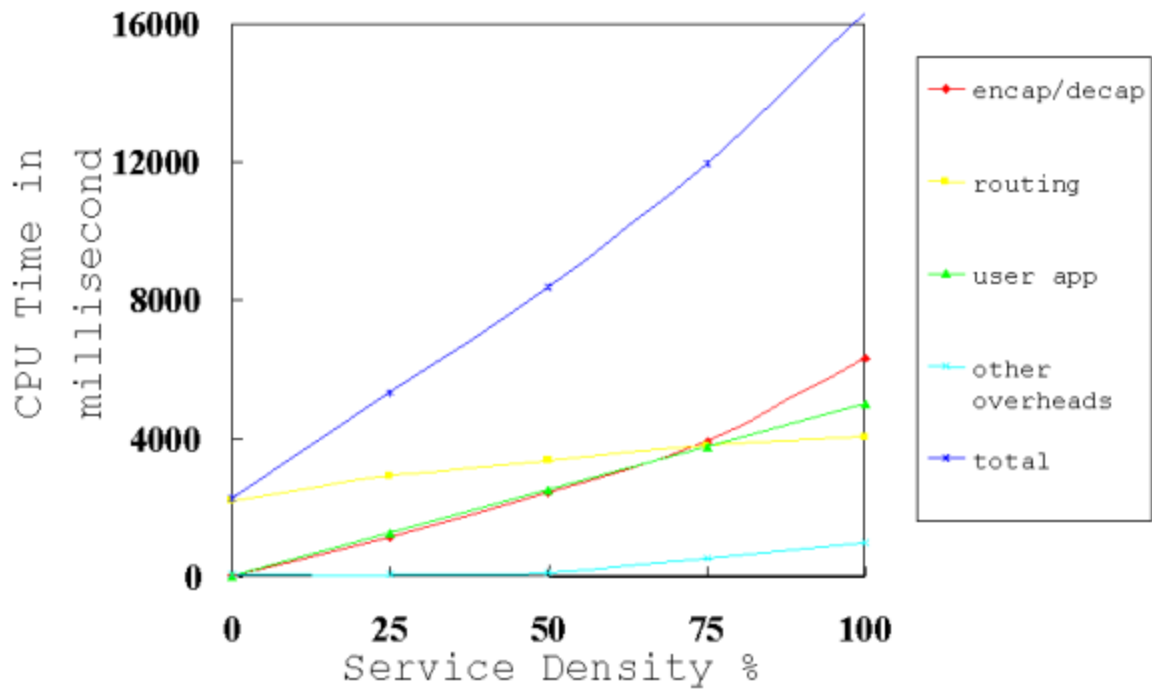


Fig 4.1 CPU Time Cost in EDIF Service Mode

In Fig 4.2, we can see that routing takes almost all CPU time used by the adaptation router. The percentage decreases as service density grows, because the time used by user applications and encap/decap of network capsule grows faster than that by routing.

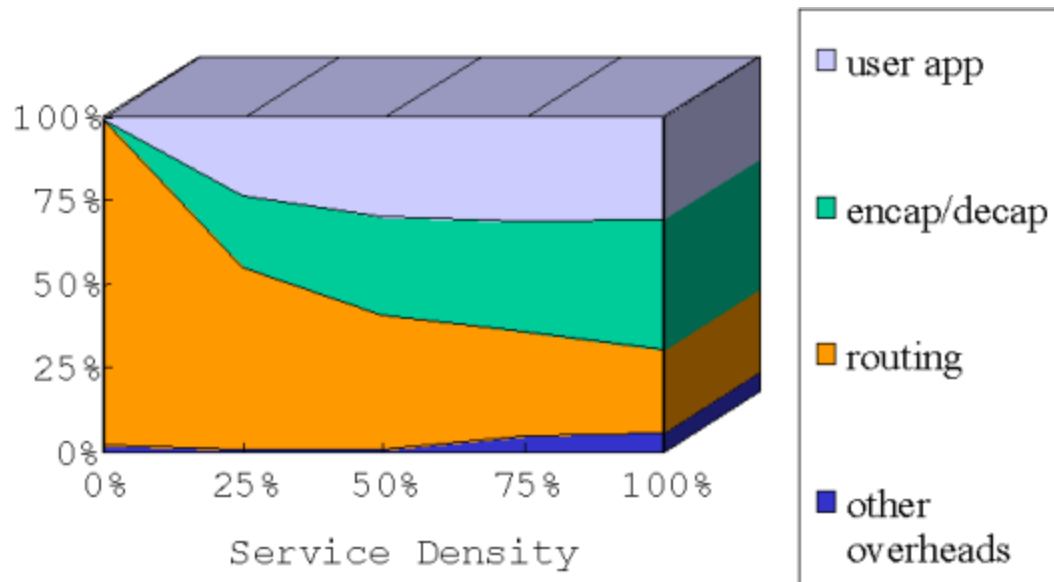


Fig 4.2 Relative Percentages of CPU Time Cost in EDIF Service Mode

4.3.2 CPU usages for FSF service:

In this section we provide the performance of the Full Search Filtering (FSF). We still send total 5M bytes data through the adaptation router with variety of service densities. Fig 4.3 plots the absolute value of CPU time cost by FSF schemes, and Fig 4.4 plots the relative CPU cost percentages for each component.

We can see from those graphs that the CPU time used by most components almost remains the same regardless of the change of service density. CPU time used by user applications does increase a little bit while the service density increases. This is due to streams in service range will be adapted after targets have been found. In this test, the adaptation is to substitute the keyword with a hyper link. The higher service density, the

more actions will be taken, and thus more CPU time will be consumed. However, user applications spend most of their CPU resource in sequential searching the keyword. The substitution only takes little CPU time. That's why the user application CPU does not increase much.

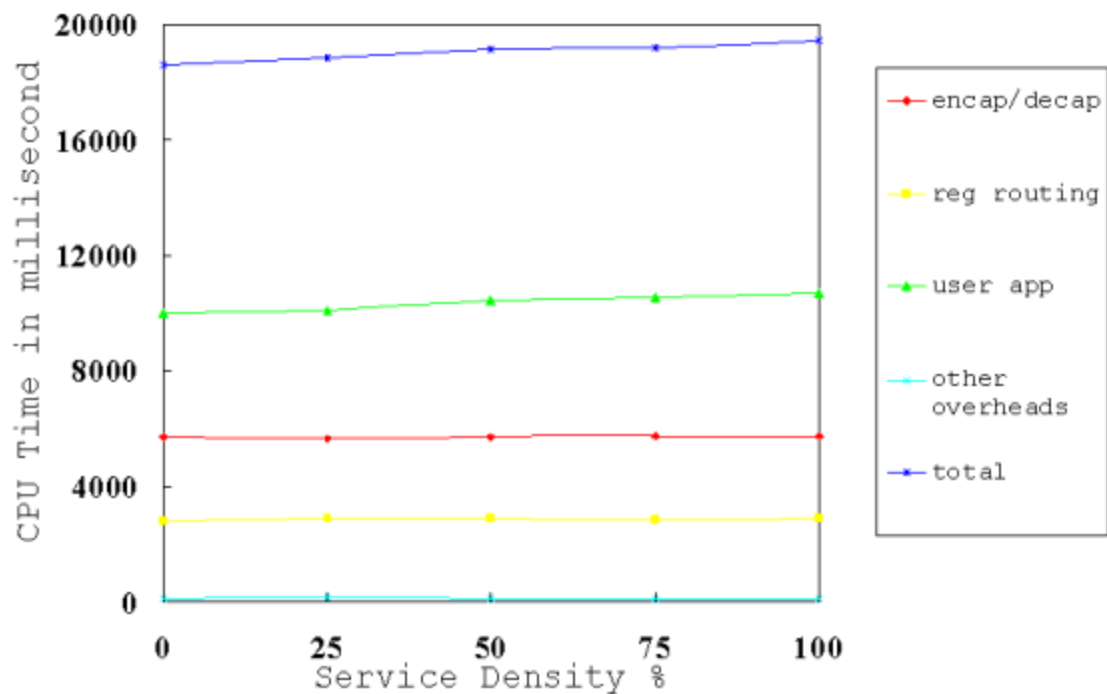


Fig 4.3 CPU Time Cost in FSF Service Mode

If we compare Fig 4.4 with Fig 4.2, we will find out one of the reasons why EDIF mode is much faster than FSF mode when service density is low. In FSF mode, when the service density is low, the user application and encap/decap procedures take more than 80% in relative CPU cost, which is totally unnecessary in EDIF mode. When the service density grows, the time for user application and encap/decap procedures grows naturally in EDIF mode, but we'll see the relative CPU cost for user application is still less than

that of FSF mode. This is because sequential search, which has $O(n)$ complexity, is used in FSF mode, while EDIP enabled indexing search, which has $O(1)$ complexity, is used in EDIF mode.

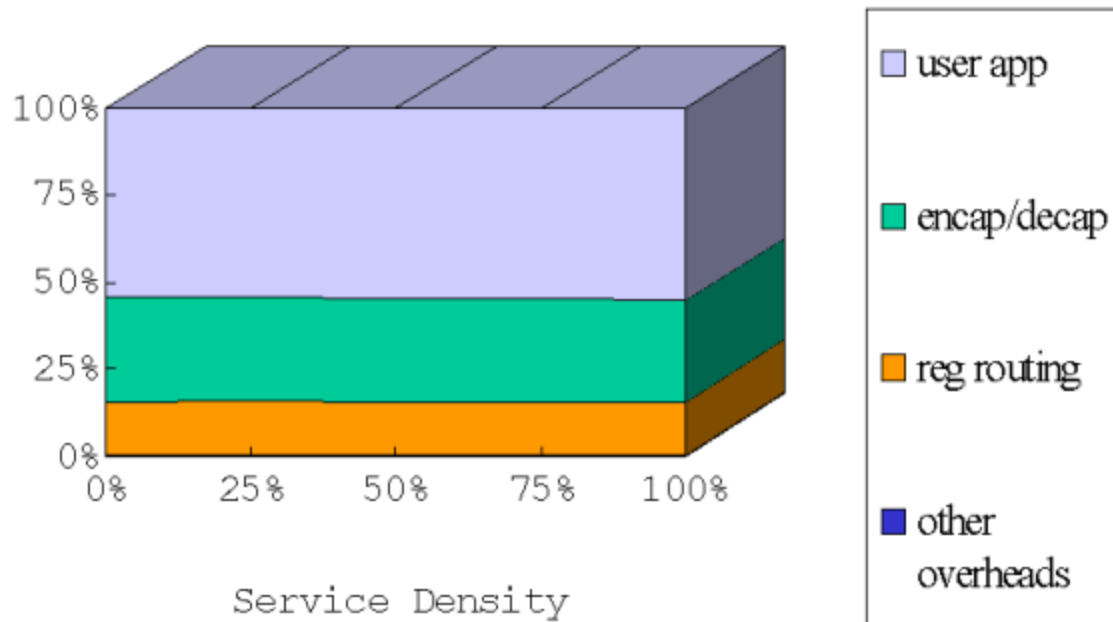


Fig 4.4 Relative Percentages of CPU Time Cost in EDIF Service Mode

4.3.3 CPU Time Comparison among EDIF, FSF and NR Mode

In this section we put performance data from EDIF and FSF mode together, plus the CPU time used by a simulated normal router without any service. The performance of NR mode represents the best possible performance we can achieve.

As Fig 4.5 shows, the EDIF incurred much smaller cost than FSF throughout. Particularly interesting is the points with a low service density. Here simple FSF incurred a cost about 14 times higher than that of a normal router. However, the EDIF performs

almost as good as the normal router. This is because of two reasons: (1) the marking mechanism allows EDIF to avoid decapsulations and encapsulations; (2) no sequential searching happens. In contrast the FSF has to decapsulated the entire stream and sequentially search whether there is a serviceable packet or not. Naturally, with the increased service density, the cost of service is increased in both the schemes. Notably, even when service density=100%, the EDIF mechanism could perform better. This is because EDIP header enabled index searching is much faster than sequential search that FSF must take.

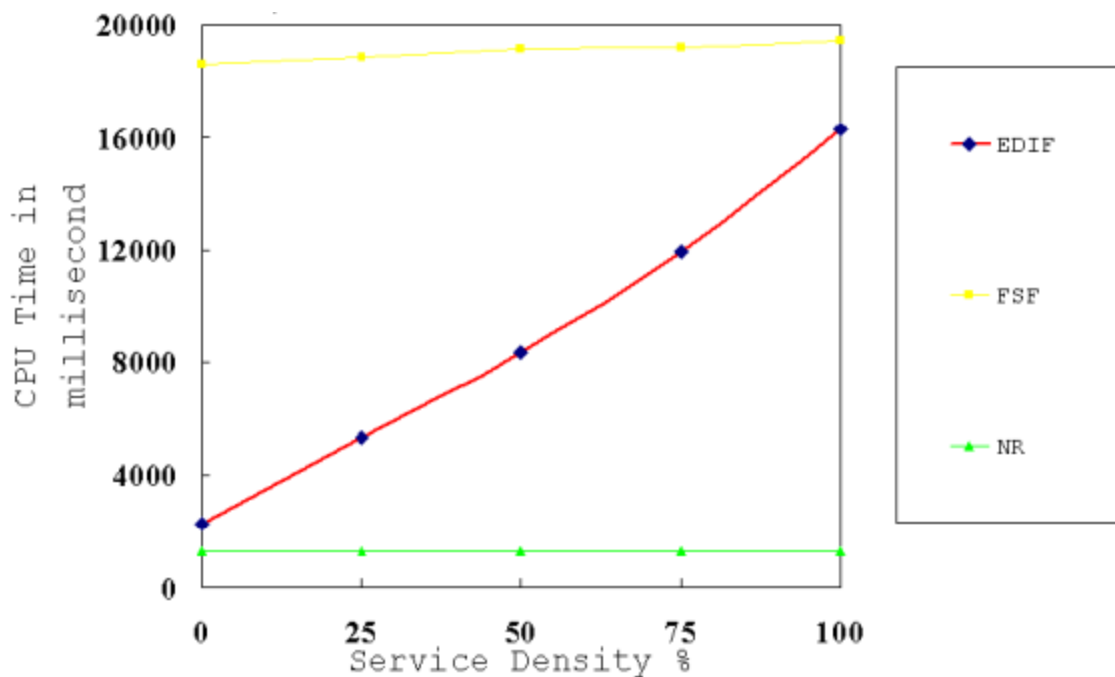


Fig 4.5 CPU time comparison among EDIF, FSF and NR mode

4.3.4 Throughput Comparison among EDIF, FSF and NR Mode

CPU time is not all for a system. People may concern more for overall performance than CPU time cost only. The difference here is, the system spend considerable amount of resource on system calls, which is not counted in previous comparisons. System calls include receiving a package from network, sending a package to network, print messages on the screen and so on. Most system calls are inevitable (such as receiving/sending IP packages), and they take significant amount of system resources in all of the three service modes. This fact neutralizes some benefit we gain from less CPU cost in EDIF mode. However, as shown in Fig 4.6, EDIF remains strong when service density is low (over 50% higher throughput when service density close to 0), and still better than FSF even when service density is 100%.

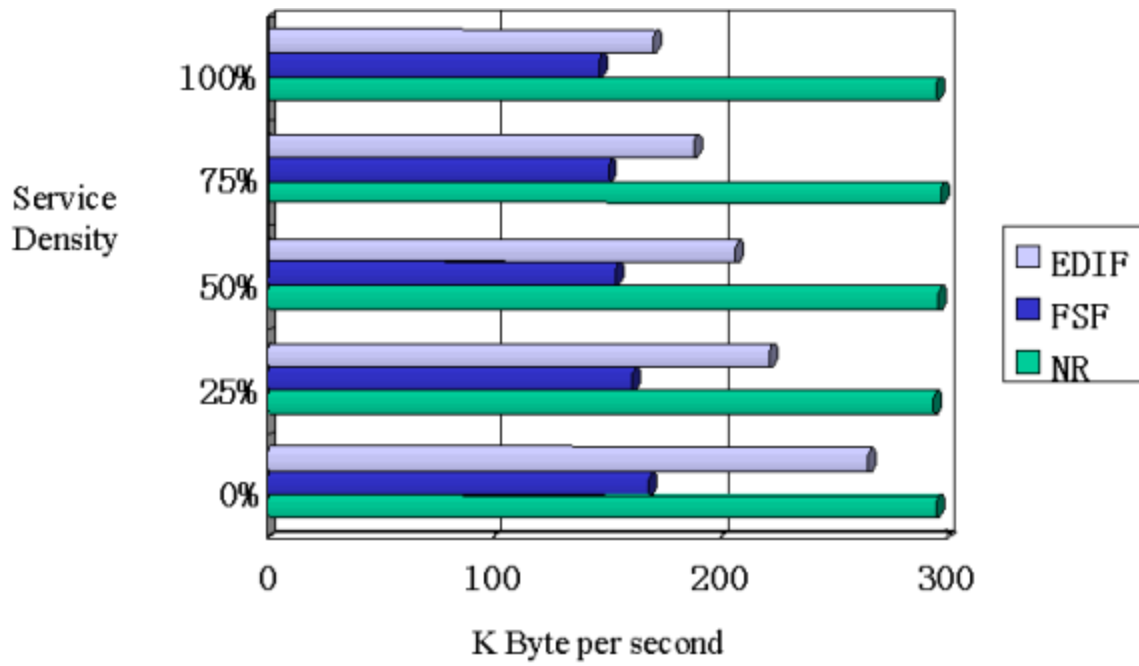


Fig 4.6 Throughput comparison among EDIF, FSF and NR mode

4.3.5 Average Package Delays

If one can say the adaptation server administrator cares more about the system throughput, what the end users concern more is the package delays. No one wants to view a web page half a minute later after (s)he clicks a link. Although real delay time for an end user also depends on network conditions, we recorded incoming and outgoing time for packages flow through our experimental adaptation router. Then we average the difference between each pair as the average package delay on our adaptation router. Fig 4.7 plots the result. We can see that EDIF mode is almost as good as NR mode when service density is low, while it still has advantages over FSF mode even when service density grows to 100%.

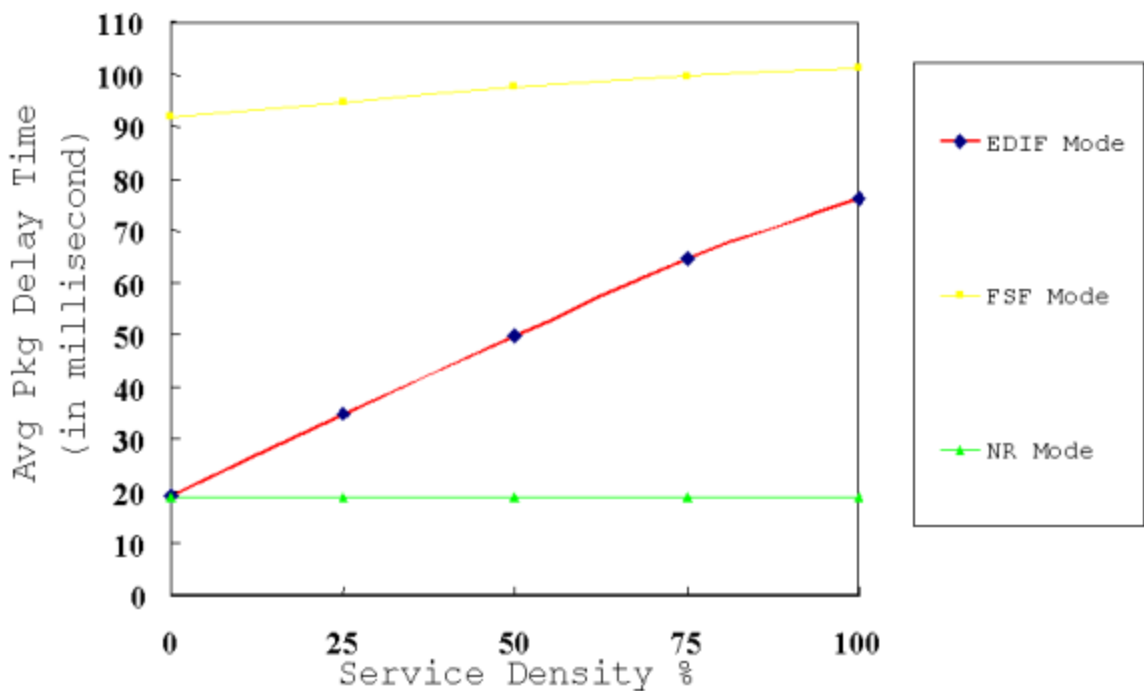


Fig 4.7 Average Packages Delays

4.4 Further Analysis

4.4.1 Space Cost

In this analysis, we inspect the space cost of EDIP headers. We assume a stream with EDIP header is divided into $pknum$ IP packages; each one is $pksize$ bytes in size. We also assume that there are n keywords inside this stream, and the i^{th} keywords will appear at the frequency of f_i times/byte, which we call key density of the i^{th} keyword.. The frequency for any keyword that will appear is F time/byte. Let A_i be the total number of indexes, and B_i be the total number of key blocks for the i^{th} keyword. Since each key block contains at least one index for a key, we have $B_i \leq A_i$. S_{index} denotes the size of an index in bytes. Since we use 16 bits to express an offset (an index), $S_{index}=2$. S_{kbr} denotes the size of a key block in bytes, excluding the indexes. We have $S_{kbr}=4$. S_{gf} denotes the size of a general field in bytes, and $S_{gf}=12$. S_{extra} is the extra space needed for EDIP headers, $S_{original}$ is the size of the original stream. E denotes the extra percentage of space needed to accommodate EDIP headers. We summarize the discussion with equations from EQ 4.1 to 4.6. The result of E is expressed in EQ 4.7:

$$F = \sum_{i=1}^n f_i \quad (\text{EQ 4.1})$$

$$A_i = f_i \times pksize \times pknum \quad (\text{EQ 4.2})$$

$$B_i \leq A_i \quad (\text{EQ 4.3})$$

$$S_{extra} = \sum_{i=1}^n \sum_{j=1}^{A_i} S_{index} + \sum_{i=1}^n \sum_{j=1}^{B_i} S_{kbr} + \sum_{i=1}^{pknum} S_{gf} \quad (\text{EQ 4.4})$$

$$S_{original} = pksize \times pknum \quad (\text{EQ 4.5})$$

$$E = \frac{S_{extra}}{S_{original}} \quad (\text{EQ 4.6})$$

$$E \leq F \times (S_{index} + S_{kbr}) + \frac{S_{gf}}{pksize} \quad (\text{EQ 4.7})$$

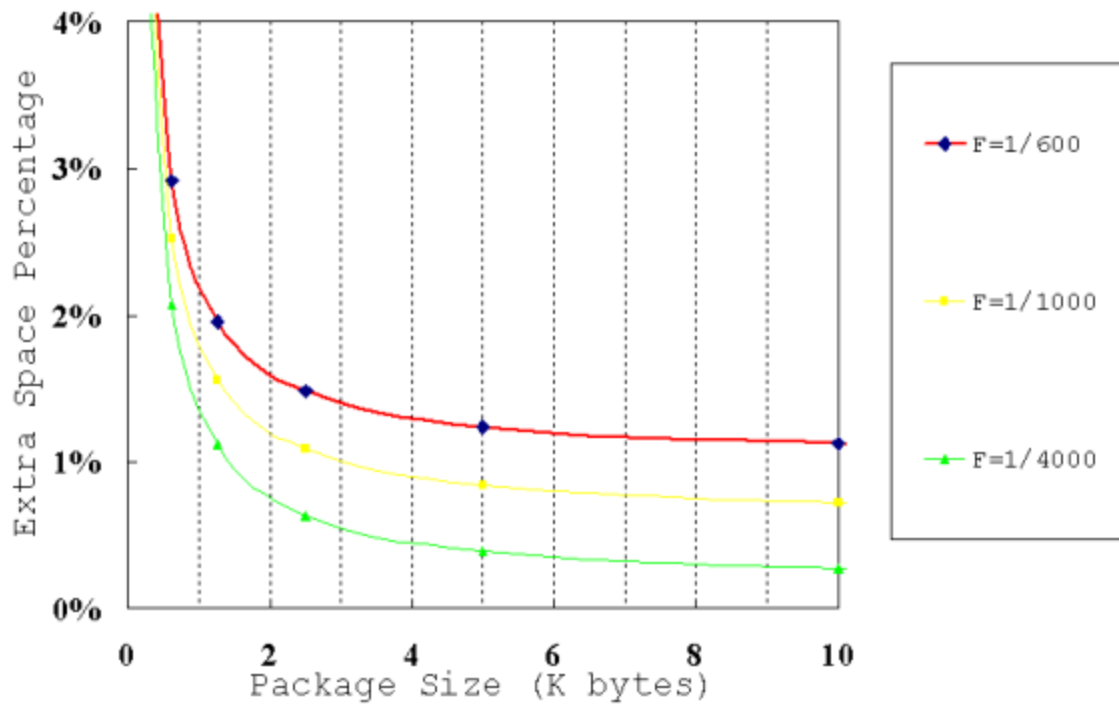


Fig 4.8 Extra Space Percentage Caused by EDIP Headers

The result for EQ 4.7 is plotted in Fig 4.8. Here $S_{index}=2.$, $S_{kbr}=4,$ and $S_{gf}=12.$ The extra space cost is below 2% when the average package size is larger than 1500 bytes², even if $f=1/600,$ which is considered a very high key density³. When the package size is larger than 5k bytes, the extra space percentage will remain small, but will not be significantly reduced. Note that we assume the service density is 100%, which means each stream has EDIP headers and it is the worst case. In real world, service density should be kept low, which makes even lower the extra space percentage for EDIP headers.

²MTU: Maximum Transmission Unit, which is the largest size frame, specified in bytes, that can be sent in a frame-based network. Normally, IP packages with content are divided into multiple frames to transmit via network. 1500 byte is the suggested value of MTU size by most ISPs.

³In practice, key density can always be controlled, by using fewer gross keys and application level processing to find the real keys.

4.4.2 Impact on Different Processor Speeds

It is important to note that in the entire operation the computation is performed in three levels. These are (i) IP level (stream controller, routing, buffering, etc.), (ii) intermediate level (encapsulation, decapsulation, etc.) and (iii) application level (searching/indexed jump, filtering, etc.). In order to examine the computational complexity, we use $C_{Component}^{ServiceMode}$ to denote the *Complexity* for *Component* in *ServiceMode*. For example C_{act}^{EDIF} denotes the complexity of user actions (adding links to web page in active hyperlinking example) in EDIF service mode. Note that in EDIF service mode, the complexities for routing⁴ are different for IP packages in service range from those out of service range. Suppose $P\%$ is the service density, which means how many percentages of IP packages are in service range. EQ 4.8 and EQ 4.9 describe the total complexity for EDIF service mode and FSF service mode, respectively.

$$C^{EDIF} = P\% \times (C_{act}^{EDIF} + C_{search}^{EDIF} + C_{encap}^{EDIF} + C_{decap}^{EDIF} + C_{routkey}^{EDIF}) + (1 - P\%) \times C_{routnokey}^{EDIF} \quad (\text{EQ 4.8})$$

$$C^{FSF} = P\% \times C_{act}^{FSF} + C_{search}^{FSF} + C_{encap}^{FSF} + C_{decap}^{FSF} + C_{routing}^{FSF} \quad (\text{EQ 4.9})$$

However, the average time to process a package is not only decided by processing complexity, but also by the speed of processor. We assume PP_{APP} , PP_{INTER} and PP_{IP} are

⁴Here “routing” includes two tasks: (1) choosing paths among nodes in network (regular routing) (2) choosing the paths inside adaptation router (routing among buffers, stream controllers)

the power of processors for user application level, intermediate level and IP level, respectively. EQ 4.10 shows the average time to process an IP package in EDIF service mode and EQ 4.11 shows the average time to process an IP package in FSF mode.

$$T^{EDIF} = \frac{P\% \times (C_{act}^{EDIF} + C_{search}^{EDIF})}{PP_{APP}} + \frac{P\% \times (C_{encap}^{EDIF} + C_{decap}^{EDIF})}{PP_{INTER}} + \frac{P\% \times (C_{routkey}^{EDIF} - C_{routnokey}^{EDIF}) + C_{routnokey}^{EDIF}}{PP_{IP}} \quad (\text{EQ 4.10})$$

$$T^{FSF} = \frac{P\% \times C_{act}^{FSF} + C_{search}^{FSF}}{PP_{APP}} + \frac{C_{encap}^{FSF} + C_{decap}^{FSF}}{PP_{INTER}} + \frac{C_{routing}^{FSF}}{PP_{IP}} \quad (\text{EQ 4.11})$$

In the tests we conducted in previous several sections, we use a single processor for all three levels of processing, which means $PP_{APP} = PP_{INTER} = PP_{IP}$. However, in reality, we could use RISC technology in lower levels, which contains simpler instructions. Some vendors made chips dedicated to routing or IP decoding/encoding, which makes $PP_{IP} > PP_{INTER} > PP_{APP}$. For simplicity, we assume that each lower level is a ($a \geq 1$) times faster than its immediate higher level. We'll have EQ 4.12:

$$PP_{IP} = a PP_{INTER} = a^2 PP_{APP} \quad (a \geq 1) \quad (\text{EQ 4.12})$$

Let R_{FSF}^{EDIF} be the ratio of speed of EDIP over FSF service mode. From EQ 4.10, EQ 4.11 and EQ 4.12, we'll get EQ 4.13, shown as following:

$$R_{FSF}^{EDIF} = \frac{T^{FSF}}{T^{EDIF}} = \frac{P\% \times C_{act}^{FSF} \mathbf{a}^2 + [C_{search}^{FSF} \mathbf{a}^2 + (C_{encap}^{FSF} + C_{decap}^{FSF}) \mathbf{a} + C_{routing}^{FSF}]}{P\% \times [(C_{act}^{EDIF} + C_{search}^{EDIF}) \mathbf{a}^2 + (C_{encap}^{EDIF} + C_{decap}^{EDIF}) \mathbf{a} + C_{routkey}^{EDIF} - C_{routnokey}^{EDIF}] + C_{routnokey}^{EDIF}}$$

(EQ 4.13)

For a given system, \mathbf{a} and all complexities are fixed values. For example, in our tested system, we have $\mathbf{a}=1$. The complexities values can be derived from the data of Fig 4.1 and Fig 4.3. For example, in our experimental system, we have:

$$C_{act}^{FSF} = 7 \qquad C_{act}^{EDIF} + C_{search}^{EDIF} = 50$$

$$C_{search}^{FSF} = 100 \qquad C_{encap}^{EDIF} + C_{decap}^{EDIF} = 61$$

$$C_{ecap}^{FSF} + C_{decap}^{FSF} = 57 \qquad C_{routnokey}^{EDIF} = 22$$

$$C_{routing}^{FSF} = 29 \qquad C_{routkey}^{EDIF} = 40$$

Fig 4.9 now plots the EDIP scheme's relative speedup for three different speed differentials ($\mathbf{a}=1,2$, and 3). We could see that the speed differential among these three levels can significantly affect the overall performance of the system. As can be noted that advantage of EDIP increases with large α . The time saving is particularly dramatic if service density remains small.

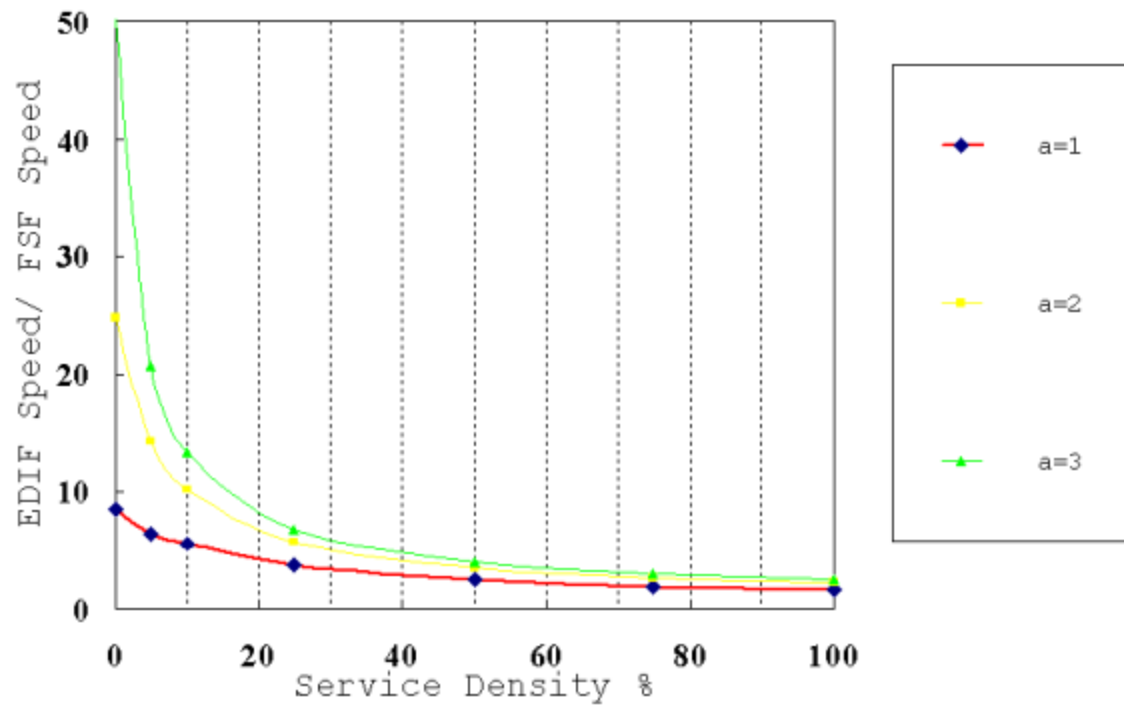


Fig 4.9 EDIF's Speedup Over FSF Model with Different a Value.

CONCLUSION

Fast intercept of streamed data is a growing concern in networking. The application level embedded processing is rapidly increasing and can be a potential bottleneck in Internet traffic carriage. The network protocols and packet data structures have been designed mostly for end-to-end processing. In this thesis we have presented a part of our research, which looks into mechanisms that can provide scope and indexing information to intermediate network hubs to enable random access in a stream. As shown in the experiment, these mechanisms expedite process of passing stream significantly under common conditions. And with potential hardware acceleration, much better performance could be achieved.

Another important provisioning is the sharable and mobile code servers inside network. In recent years some advances have been made in programmable networks. Among them active networks [21][22][23] initiative proposes the generalization of the traditional router concept— where transiting packets can be modified almost in any way with custom embedded program modules in the network elements. Several other attempts are underway, where standalone processors are being added with routers.

One of the top issues here is the security. Since program codes are mobile, a local system has to make sure such “alien” codes are not harmful, no matter caused by careless

programmers or malicious cyber attackers. Although this problem can be partially solved by verification or authentication among participating parties, it seems the complete solution is designing a highly dynamic runtime environment[24][25][26].

One of the other issues is the filter programs management. When an adaptation service server is running a number of filter programs from different sources, we have to prevent them from collision and dead locks. If we look into these problems, we may find that most of them have counterpoints in a conventional OS, such as user management, sharing, deadlock prevention, security check and so on. This arose the idea of designing special purpose active network operating systems or execution environments that can systematically support adaptation services over the Internet. Several research groups are working on these issues, such as ANTS [11] and Janos Java NodeOS [15] .

We believe a commercialized fast content adaptation network will appear in near future. However, a lot of effort has to be made before such a system can be put into wide use in large scale.

REFERENCES

- [1] S. Blake, D. Black and etc., RFC 2475 “An Architecture for Differentiated Services”, 1998
- [2] “Annotation-based web content transcoding”, M. Hori, etc, The 9th WWW conference, 2000, available at: <http://www9.org/w9cdrom/169/169.htm> [Last accessed on Jun 30, 2002]
- [3] Akamai, <http://www.akamai.com> [last accessed on Jun 30, 2002]
- [4] S. J. Lee, W. Y. Ma, and B. Shen, Interactive video caching and delivery using video abstraction and summarization, Proc. International Workshop on Web caching and Content distribution (WCW'01), Jun 2001
- [5] S. Deering, R. Hinden, “Internet Protocol, version 6 specification”, RFC 2460, 1998
- [6] Wei-Ying Ma, Bo Shen and Jack Brassil, “Content Services Network: The Architecture and Protocols”, Int. workshop on web caching and content distribution, June 2001.
- [7] Oliver Spatscheck, J. S. Hansen, J. H. Hartman and L. Peterson; Optimizing TCP forwarder performance, IEEE/ACM Trans. Networking 8, 2, Apr. 2000, pp146 - 157.

- [8] D. Maltz and E Bhagwat, "TCP splicing for application layer proxy performance," IBM, <ftp://ftp.cs.cmu.edu/user/dmaltz/Doc/splice-perf-tr.ps>, Mar. 1998.
- [9] Open Pluggable Edge Service (OPES), <http://www.ietf-opes.org> [Last accessed on Jun 30, 2002]
- [10] "A Model for Open Pluggable Edge Services", G. Tomlinson, etc.,
draft-tomlinson-opes-model-00.txt
- [11] ANTS <http://www.cs.washington.edu/research/networking/ants/>
- [12] "Developing Web Applications for Pervasive Computing Devices", Steve Imes,
IBM webserver studio document, Jan 2001
- [13] "Ubiquitous Internet Application Services on Sharable Infrastructure", Javed I. Khan and Yihua He, technical report, available at:
<http://bristi.facnet.mcs.kent.edu/~javed/medianet/techreports/TR2002-03-02-asp-KH.pdf>
- [14] "OPES Architecture for Rule Processing and Service Execution", Lily Yang, Marcus Hofmann, draft-yang-opes-rule-processing-service-execution-00.txt
- [15] Janos Java NodeOS <http://www.cs.utah.edu/flux/janos/jnodeos.html> [Last accessed on Jun 30, 2002]
- [16] Spyglass-Prism. <http://www.spyglass.com>. Last accessed on Jun 30, 2002

- [17] J. Smith, R. Mohan, and C. Li, "Scalable multimedia delivery for pervasive computing," *ACM Multimedia*, 1999.
- [18] Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer, Adapting to network and client variation using active proxies: lessons and perspectives, *IEEE Personal Communication*, Vol. 5, No. 4, pp. 10-19, August 1998.
- [19] O. Angin, A.T. Campbell, M. E. Kounavis, and R. R.-F. Liao, The Mobeware Toolkit: Programmable support for adaptive mobile networking, *IEEE Personal Communications*, Vol. 5, No. 4, August 1998, pp. 32-43.
- [20] Jeremy Elson and Alberto Cerpa, Editors, ICAP, The Internet Content Adaptation Protocol, 2001
- [21] Javed I. Khan, S. S. Yang, A Framework for Building Complex Netcentric Systems on Active Network, Proceedings of DARPA Active Network Conference and Exposition 2002, IEEE Press, San Francisco, May, 2002.
- [22] Wetherall, David, Active Network Vision and Reality: Lessons from capsule-based System, *Operating Systems Review*, 34(5): pages 64-79, December 1999.
- [23] Jonathan M. Smith, Programmable Networks: Selected Challenges in Computer Networking, *Computer*, January 1999 (Vol. 32, No. 1), pp. 40-42
- [24] S. Murphy, E. Lewis, and R. Watson: Secure Active Network Prototypes,

- Proceedings of DARPA Active Network Conference and Exposition (DANCE),
pages 166-181, May 2002
- [25] S. Krishnaswamy, J. Evans, and G. Minden: A prototype Framework for
Providing Hop-by-Hop Security in an Experimentally Deployed Active
Network, Proceedings of DARPA Active Network Conference and Exposition
(DANCE), pages 216-223, May 2002
- [26] M. Hicks, A. Keromytis, and J. Smith: A Secure PLAN (extended version), ,
Proceedings of DARPA Active Network Conference and Exposition (DANCE),
pages 224-237, May 2002
- [27] Javed I. Khan and Yihua He: Fast Intercept of A Passing Stream For High
Performance Filter Appliances, accepted by 5th International Conference on
High-Speed Networks and Multimedia Communications HSNMC'02 July 3-4
2002, Jeju Island, Korea

GLOSSARY OF TERMS AND ABBREVIATIONS

AR	Adaptation Router
AR Admin	Adaptation Router Administrator (at service provider's side)
ASDL	Active Service Distribution and Localization (Model)
CDN	Content Delivery Network
CP	Content Provider
CPI	Content Provider Initiated (Service)
CSN	Content Service Network
EDIF	Embedded Data Index Filtering (mode)
EDIP	Embedded Data Indexing Protocol
EU	End User (Agent)
EUI	End User Initiated (Service)
FSF	Full Search Filtering (mode)
GF	General Field (in EDIP header)
KB	Key Block (in EDIP header)
Key Density	The frequency of key word appearance
MA Marker	Administrator (at content provider' side)
NR	Normal Router (mode without any service)

Service Density The percentages of data volume that in service range

SMS Service Management Server

SPI Service Provider Initiated (Service)

A SAMPLE USER FILTER APPLICATION

(Active Hyperlinking)

```
int userapp()
{
    int                streamid;

    struct searchresult_t  searchresult;

    int                i;

    int                string0len, string1len;

    char    string0[]="<i>Yihua He</i>";

    char    string1[]="<i><a href=\"mailto:yihe@cs.kent.edu\">Yihua He</a></i>";

    string1len=strlen(string1);

    string0len=strlen(string0);

    for (; ; )
    {
        streamid=getstreamid();

        if (0>keysearch(streamid, 0, string0, string0len, &searchresult))

            printf("keyword not found\n");
    }
}
```

```
else
{
    printf("keyword %s found at: \n", searchresult.keyword);

    for (i=0; i<searchresult.numofindex; i++)
        printf("%d  ", searchresult.offset[i]);

    printf("\n");

    for (i=searchresult.numofindex-1; i>=0; i--)
    {
        temp=searchresult.offset[i];

        deletebyte(temp, string0len);

        insertbyte(searchresult.offset[i], string1, string1len);
    }
}

sendstream(streamid);
}

return(0);
}
```

A SAMPLE MARKER PROGRAM AT SOURCE SIDE

```
int marker1 (char *in, int in_len, struct searchresult_t *searchresult)
{
    char kwd[]="<i>Yihua He</i>";

    int    offset;

    strcpy(searchresult->keyword, kwd);

    searchresult->keylen=strlen(kwd);

    searchresult->numofindex=0;

    for (offset=0; offset+strlen(kwd)<in_len; offset++)
    {
        if (bcmp(in+offset, kwd, strlen(kwd))==0)
        {
            searchresult->offset[searchresult->numofindex]=offset;

            (searchresult->numofindex)++;

            if ((searchresult->numofindex)>=MAXNUMOFFFOUND)
            {
                printf("too many keyword found! \n");

                return (-1);
            }
        }
    }
}
```

```
        }  
    }  
}  
  
return (searchresult->numofindex);  
}
```