Proceedings of the IEEE/WIC International Conference on Web Intelligence, WI'03, October 2003, Halifax, Canada, pp.579-583

Embedded Data Indexing for Fast Stream Interception by Internet Appliances

Javed I. Khan & Yihua He
Media Communications and Networking Research Laboratory
Department of Computer Science, Kent State University, Ohio, USA
javed|yhe5@kent.edu

Abstract

Interception of a data stream is central to any intelligent and dynamic processing of web information. It is perhaps as fundamental to Internet services' overall architecture as the design of disk scheduling to the conventional machine architecture. In this paper we discuss an IPv6 based indexing protocol that can facilitate random access into multilevel hierarchically encoded content streams and provide serious performance boost to web content processing.

Keywords: stream interception, filtering, network appliances, IPV6.

1. Introduction

Content interception and processing is becoming a common task in Internet. We are already seeing emergence of a variety of internet appliances. Beginning from cache, proxy, filters, firewalls and gateways, there are now host of new services including content adaptation, content personalization, location-aware data insertion, to virus filters [1,2,5,6] -- all are fundamentally stream interception machines requiring some form of intermediate access inside transiting traffic's content. However, if we look into the designs of current transport technology, we will see that it has little facility that enables efficient stream intercept.

Most real-life content, particularly high performance networked multimedia transport data carried over a network packet are multi-level hierarchically encapsulated (that means bags within suitcases). For example an H.261 video element may require access to tens of pre-dependent elements before the element of interest can be read. The situation is also difficult for recent tag-delimited content protocol standards (such as XML, HTML), where this dependency is formally infinite. 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. The problem has major implication on the CPU cycle, memory size, and overall performance of any intercepting appliance system's architecture. Essentially, the stream is the working data structure of a filtering system. It is perhaps as salient to Internet appliance's overall architecture as the design of disk scheduling algorithm or multilevel memory/cache organization is to the conventional machine architecture. While, simple end-to-end applications can 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 key factor for high performance stream data processing appliances.

Recently, a number of investigations have been made in the broader area of provisioning internet services. Chan [1] presents a graph oriented model of distributed web processing. The Open Pluggable Edge Services (OPES) [9] working group built up a rule-based specification transmitting scenario [10, 14] to express the service requests paving the way for growth of internet services. ASDL [13] model considered various deployment models and classes of filtering services. 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 filtering. Akamai [3], ESI [4], and WebSphare [12] use application level XML-like markup language to define web page components for dynamic assembly and delivery of web applications at the edge of Internet. However, full search is performed at the end point to redetect the markers. No work exists that focused on provisioning random access into content leading to overall speedup of appliance's performance.

In this research we focus on this important and yet unexplored problem of facilitating fast access into hierarchical multi-protocol based internet content transport. Recently, in [11] we have discussed an architecture for Internet Appliance Server. In this paper we present our vision towards a logical design of a content indexing protocol that can be wrapped around a TCP stream to facilitate dynamic index based pseudo-random access. We propose an IPv6 extension header based scheme. Even without any kernel level support, we show the potential of realizing about 500-800% speedup over today's content servicing technique in normal conditions with such schemes.

2. Approach

2.1 Models of Appliance Services

Before we discuss the scheme, first let us consider the generalized model of appliance services. In this 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 *appliance* processing (AP) *machines* called *appliance server* (AS) 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 AS servers.

The application services can be classified in various ways under this model. In special cases of *backend filtering* the CP and AP can be collocated in content provider's site. A special case of AP intervention is the *passive filtering* service where appliance program 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 AS server). Complex application service can be composed by *service piping*.

2.2 Content Encoding

The provisioning of pseudo random access to the content by the AP requires implementing some form of indexing data structure conceptually similar to *file allocation tables* used in file systems. This requires consideration about the organization content stream.

Unlike conventional file systems internet content is coded with multilevel hierarchical protocols. The content encoding is performed at various levels of transport. Most low level network protocols are performance oriented and their elements (or fields) are of fixed length (such as Ethernet or IP address fields). Some have variable length. Many also have conditional elements. Some complexity arises due to extensive amount of conditional fields (such as MPEG). Also the fields are sequence sensitive and thus element names are implicit. On the other hand, most upper level transports protocol elements are designed for richer and flexible knowledge encoding and are tag delimited (such as HTTP or HTML) rather than of fixed length. XML further allows dynamic definition of protocol elements. During the actual network transport, TCP or UDP provides a stream abstraction to the application processing unit. Beneath it the data is actually 'paged' as IP packets with variable sized chunks. Technology such as ATM can add further paging. From the ASP context however, a particular AP may not require all protocol fields, can add new or delete existing fields. Typically, any access to an element requires access to all elements of all parent protocols as well as access to sequence sensitive elements of protocol in the same level.

2.3 Scopes for Speedup

In this context, the operation of and application filter processing can be expedited by two techniques. The first is pre-indexing the content stream and allowing indexed direct access into to the stream. Second is the selective decapsulation reencapsulation of only the pertinent data segments. Generally any filtering operation is conditional. The processing has a specified "*scope segment*" and a "*key segment*" in it. The scope segment offers the opportunity to avoid unnecessary decapsulation/ encapsulation at the intercepting server. We have developed the *Embedded Data Indexing Protocol* (EDIP), schema that can exploit the above.

3. Proposed EDIP Indexing Mechanism

3.1 EDIP Extension Header Format

While several data structures can be used for index table, we propose a flat organization of the table where each entry will include a key (protocol element) identifier and a set of offset fields. Also we organize the table in per IP packet basis using the IPv6 extension mechanism. The IPv6 packet is considered the "scope segment" for the keys listed in its EDIP index table. Fig-1 shows the proposed IPv6 extension header based EDIP indexing scheme. Essentially each IPV6 packet containing filtering data can carry the table. 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

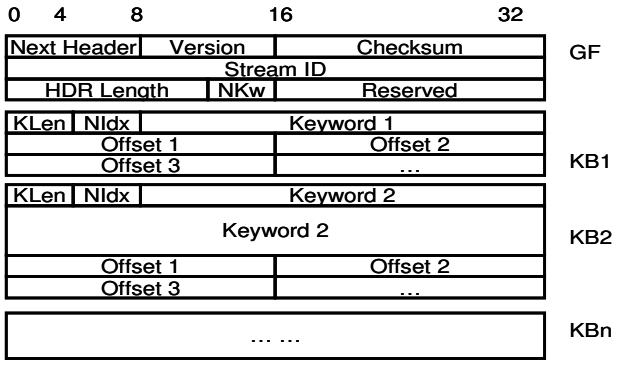

Fig -1: EDIP Extension Definition

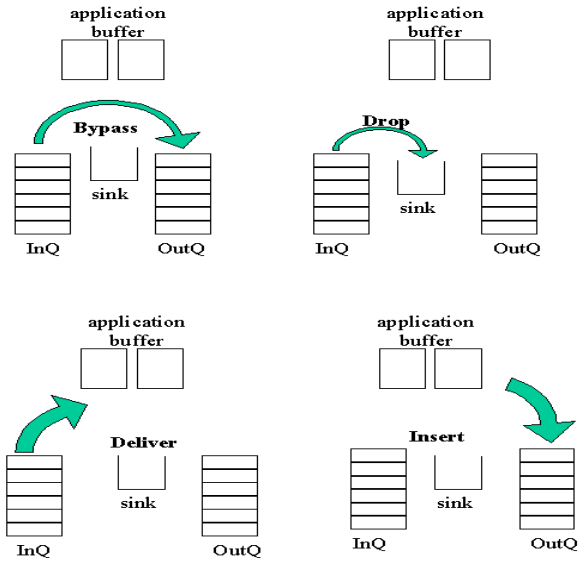

Fig-2: Data Manipulate API Operations

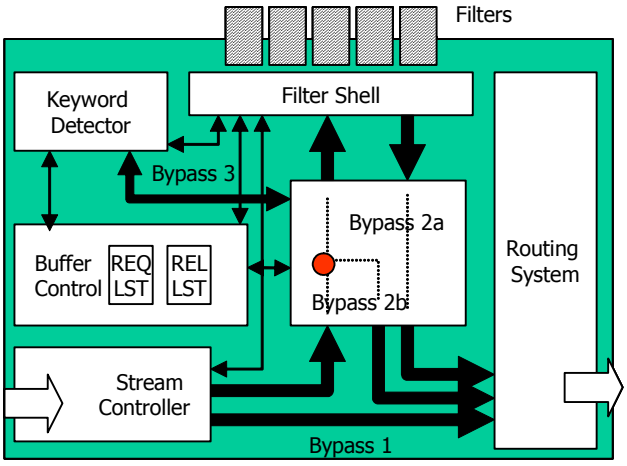

Fig-3: Logical Architecture of Appliance Server

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.

3.2 Application Processing

The application filters also is given a set of special services APIs to take advantage of the EDIP indexing. We have designed the *Stream Edit API* set. These APIs can be viewed as two parts: (1) the service administrative APIs and (2) the stream manipulate APIs. The filter program can use the service administrative API subset to (i) activate and deactivate various marking and detection steps, (ii) register and deregister keyword list which it needs and (iii) receive invocation along with the byte offset of requested keywords, when they arrive. The stream edit API is the set which with a filter receives data and edits the content stream. Fig-4 explains the stream edit operations. These operations has been defined within the context of a paired incoming (InQ) and outgoing (OutQ) transport (TCP in this case) socket streams. The application creates sockets in usual way. The edit operation begins by associating an InQ with an OutQ. The bypass operation allows a specific number of application bytes to be moved from InQ to OutQ by the embedded layers (TCP, IP, and the router intercept in this case) directly. It can also request the enhanced TCP layer to drop specific segments of bytes. When the application needs the data it can request segments of stream to be delivered to the application buffer. It can also insert bytes into a stream from the application buffer.

3.3 Internet Appliance Server

Given the EDIP definition, it is possible to propose multiple appliance server architecture that can support the indexed intercept and corresponding stream edit operations by the filters. Fig-3 presents the logical architecture of the Appliance Server. The appliance intercepts IP stream and forwards and outgoing packets to a normal routing system. It has three main logical components are (i) stream controller, a (ii) keyword manager and (iii) a EDIP buffer controller. Essentially, these units implements four flows. Bypass-1 enables the normal IP packets to be forwarded directly to the router. Bypass-3 is for key segments where the content containing key. These are always encapsulated and processed. Bypass-2 is for scope segment data which are conditionally encapsulated based on the condition. Among these bypass 2a is taken by the scope data which requires actual filtering. The logical operations given in Fig-4 ensure the proper flow of content segments in these paths. A stream controller's input is incoming IP packets. It forwards those IP packages without EDIP towards the forwarding router, but intercepts and stores those with EDIP header into the EDIP Buffer for further actions. Further more, if the EDIP header contains any key block, the stream interceptor decrypts it with corresponding decryption key, and sends the keywords, contents and stream IDs to Keyword Detector. The buffer controller maintains two lists --- a *required_steamid_list* and a *release_streamid_list*. The buffer controller checks if there are any IP packages with the stream id listed in the two lists. Those in the required list should be sent to

application level and those in the release list should be

```
If (EDIP header doesn't exist)
    Send IP package to regular routering; //1

If (EDIP header exists) {
   if (EDIP header doesn't contain any key blocks)
        Put IP package in the buffer;  //2
   if (EDIP header contains some key blocks)
        Decode the EDIP header with decryption key;
        Send keywords contents and  streamid to
              Detector; //3
        Put IP package in the buffer; //and 2
   }
```

Fig 4(a) Stream Controller Operation Logic

```
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; //2a
   If ( IP package's stream id is listed in release_stream_id[])
         Send IP package to regular routing ;  //2b
    If ( IP package's timestamp expired)
         Send IP package to regular routing; //2b
```

Fig-4(b) Buffer Controller Operation Logic

```
KeywordDetector()
    GetFromIP(I_Keyword, I_Content, I_Streamid);
    If (! (I_Keyword in Keyword_list))
         release(I_Streamid);
    else{      if (I_Content matches corresponding
         entry in condition_list)
              require(I_Streamid);
              else  release(I_Streamid);
   }
```

Fig 4(c) Keyword Detector Operation Logic

forwarded towards their destinations. For efficiency reason every IP packages in the buffer should have a timestamp. If timestamp expires, the IP package should be released, whether processed or not. The keyword detector is checks if the keywords sent by stream controller are in the active keyword list. If not, the stream id should be added into *release_streamid_list* in the buffer controller. If yes, the stream id can be added to the *required_streamid_list*.

3.4 EDIP Index Generation

Several approaches have to be accommodated to generate the EDIP index based IPv6 framing. Though these need not to be the part of EDIP definition. To extract the index table under co-operative filtering the application service provider can also sends a *marking servlet* to the CP server for marking of the content stream. In stealth scheme, this would require provisioning a string processor type servlet to be run prior to the appliance server.

4. Performance Analysis

We have made a full implementation of the proposed EDIF scheme at user space [11]. Here we share results from this implementation. To put the result into perspective we compare the EDIF system both with Full Search Filtering (FSF) as well as that with a Normal Router (NR). FSF represents the cost of performing the same service without the random access mechanism. For plotting we will normalize each of the costs with respect to the time it takes to process similar amount of data volume, without any service via a normal router (NR).

The first experiment, plotted in Fig-6, shows the speedup. It plots the *relative speed cost* of both EDIF, and FSF schemes. The cost of value-adding service is generally related to the amount of packets in the traffic those need the service. We call it *service density* (p). Fig-5 plots the relative speed cost for various service density. As can be seen, the EDIF incurred much smaller cost than FSF throughout. Particularly interesting is the points with a low (p). Here simple FSF incurred a cost about 16-17 times higher than that of a normal router. However, the EDIF performs almost as good as the normal router. This is because the marking mechanism allows EDIF to avoid decapsulations. In contrast the FSF has to decapsulated the entire stream whether there is a serviceable packet or not. Naturally, with the increased (p), the cost of service is increased in both the schemes. Notably, even at p=1, the EDIF mechanism could perform better. This is because the routine search can be performed at lower layer and upper layer will get index to the required data sections.

A natural concern is that how much network traffic it added to communication? Fig-6 provides the space overhead for EDIP scheme. Here we assume that service density p=1 and keywords will appear at the frequency of

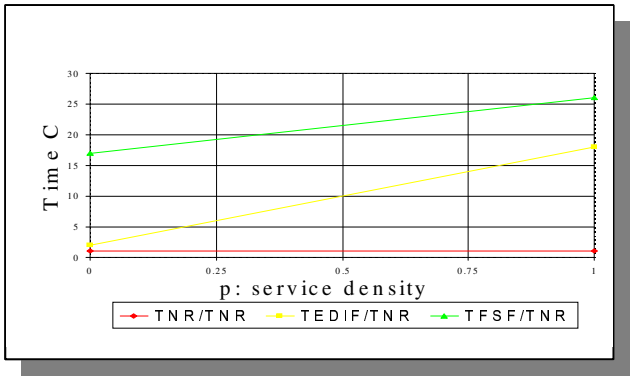

Fig-5 EDIP Speedup over Classical Filter

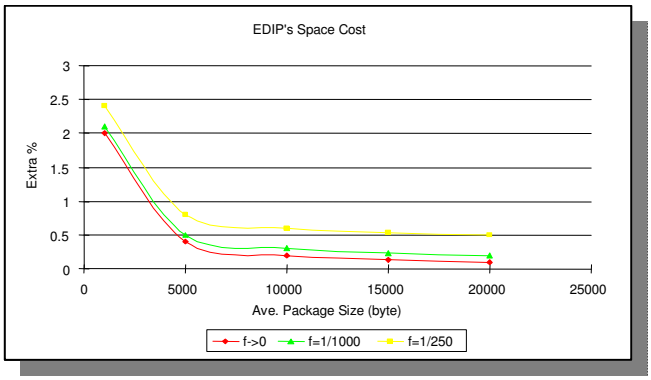

Fig-6 EDIP Space Overhead

f times/byte (assuming there is only one keyword per package, but it can appear multiple times in one packages). The result is shown in Fig 6. As evident the extra space cost is below 2% when the average package size is larger than 3k bytes, even if f=1/250, which is considered a very high frequency of keyword appearance. When average package size is over 10k bytes, the extra space cost by EDIP header is negligible.

5. Conclusions

The application level embedded processing is rapidly increasing and can be a potential bottleneck in Internet traffic carriage. The existing network protocols and packet data structures have been designed mostly for end-to-end processing without foreseeing the emergence of intermediate content processing. A stream is a working data structure for network applications. It is vital for intercepting applications to gain fast access into content. In this paper we have presented a research which investigates the issues and presents mechanisms that can provide pseudo-random access into multilevel hierarchically encoded content and speed up all such content processing applications. However, there are varieties of **content stream processing** scenarios each offering different opportunities and challenges (a classification of stream processing scenarios is in section 2.1). For example, the index creation in a non-cooperative filtering scenario can require extra overhead. However, the research in random content access will remain indispensable considering the overall growth of content processing field. It can also be noted that some form of random content access can benefit other content processing scenarios besides stream processing. Example include search tool, web-site summarization or complex machine surfing of web pages envisioned in semantic web.

Under the EDIP framework it is possible to implement the appliance server architecture in different ways offering various degrees of flexibility and performance efficiencies. As a proof of concept we have recently made a full implementation of one EDIP system (EDIP server end IPV6 Encapsulator and the interceptor server's Application Selective Decapsulator) at user space [11], including few sample filtering applications (which perform XML transcoding).

The work has been funded by the DARPA Research Grant F30602-99-1-0515 under its Active Network initiative.

6. References

[1] A. T. S. Chan, C. K. Chan, J. Cao, Programming Distributed Web Services Using WebGOP, 8th IEEE Symp. On Computers and Communications, July 2003, Turkey, pp413-418.

[2] "Annotation-based web content transcoding", M. Hori, etc, The 9th WWW conference, 2000, available at: http://www9.org/w9cdrom/ 169/169.html

[3] Akamai, http://www.akamai.com

[4] ESI, http://www.esi.org

[5] C. Y. Chang and M. S. Chen. Exploring Aggregate Effect with Weighted Transcoding Graphs for Efficient Cache Replacement in Transcoding Proxies. In Proceedings of the 18th IEEE International Conference on Data Engineering, (ICDE-02), 2002.

[6] A. Maheshwari, A. Sharma, K. Ramamritham,and P. Shenoy. TranSquid: Transcoding caching proxy for heterogenous ecommerce environments. In Proc. 12th IEEE Workshop on Research Issues in Data Engineering (RIDE '02), San Jose, CA,Feb 2002. http://lass.cs.umass.edu/papers/ps/RIDE02-trans.ps.

[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

[10] "A Model for Open Pluggable Edge Services", G. Tomlinson, etc., draft-tomlinson-opes-model-00.txt

[11] Javed I. Khan & Yihua He, Fast Intercept of a Passing Stream for High Performance Filter Appliances, 5th IEEE International Conf. on High Speed Networking and Multimedia Communications, HSNMC2002, Jeju, Korea, July 2002., pp122-127.

[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

Proceedings of the IEEE/WIC International Conference on Web Intelligence, WI'03, October 2003, Halifax, Canada, pp.579-583