Data Integrity Related Markup Language and HTTP Protocol Support for Web Intermediaries[*]

Chi-Hung Chi[1], Lin Liu[1], Xiaoyin Yu[2]
[1] School of Software, Tsinghua University, Beijing, China 100084
[2] School of Computing, National University of Singapore, Singapore
Contact email: chichihung@mail.tsinghua.edu.cn

Abstract. In this paper, we propose the markup language framework and its associated HTTP protocol extension to support data integrity for active web intermediaries. There are three aspects that our language framework would like to focus on. Firstly, the content server can specify its intended authorization and task to multiple web intermediary proxies in its content objects easily. Secondly, a web intermediary performing value-added services through content adaptation needs to leave its modification footprints for possible verification by client. Thirdly, a client is facilitated to verify the received message with the server's authorizations and intermediaries' footprints without affecting his perceived web latency. We demonstrate the feasibility and practicability of this language framework through its actual implementation

1. Introduction

Recently, one key direction to provide better web quality services in heterogeneous pervasive web environment is the real-time adaptive content delivery. Basically, the research of focus is to investigate technologies and practical systems to provide more efficient and effective value-added services through real-time content adaptation in the network. Examples of these services include: image transcoding, media conversion (e.g. image to text), language translation, encoding format (e.g. lossless data compression), and local advertisement uploading.

The prosperity of the research on real-time content transformation by active web intermediaries draws great attention to the problem of data integrity. Since the same set of technologies supports the modification of a message on its way from a server to a client, independent of the authorization of the web intermediaries by the content server, how can the client trust the receiving message and how can the server ensure that what the client receives is what it intends to respond? In this paper, we would like to address this data integrity problem through the

[*] This research is supported by the funding 2004CB719400 of China.

support of markup language and HTTP protocol extension. With the complete definition of our data integrity language framework and its associated HTTP extension, we demonstrate the accuracy, feasibility and practicability of our approach through its actual implementation.

2. Related Work

There have been proposed solutions for data integrity but their context is quite different from the new active web intermediary environment here. In `HTTP/1.1`, integrity protection [7] is a way for a client and a server to verify not only each other's identity but also the authenticity of the data they send. Secure Sockets Layer [7] does a good job for the integrity of the transferred data since it ensures the security of the data through encryption. However, these methods do not meet the need of active web intermediary services because they do not support legal content modification in the data transmission process, even by the authorized intermediaries.

Recently, there are new proposals being put forward in the area of content delivery and real-time content adaptation on the web. To meet the need of data integrity for delta-encoding [4], [4] defines a new `HTTP` header "`Delta-MD5`" to carry the digest value of the reassembling `HTTP` response from several individual messages. However, this solution is proposed only for delta-encoding exclusively and is not suitable for active web intermediaries. `VPCN` [1] is another proposal to solve the integrity problem brought by `OPES`. It makes use of the concept similar to Virtual Private Networks (VPN) to ensure the integrity of the transport of content among network nodes. It also supports transformation on content provided that the nodes are inside the virtual private content network. Its main problems are the potential high overhead and the restriction of performing value-added web services by a small predefined subset of proxy gateways only. Other proposals [6] draft the requirements of data integrity solution in the active web intermediary environment. [2] proposes a `XML`-based service model to define the data integrity solution formally. However, these works are at their preliminary stages; they are just drafts or proposals without actual implementation to demonstrate the system feasibility and performance.

3. Language Support In Data Integrity Framework

In this section, we will first give the basic format/structure of the language for our data integrity framework, followed by the detailed description of how a content server can use it to express its intention to web intermediaries for content modification.

3.1. Overview

Our data integrity framework follows naturally the HTTP response message model to transfer data-integrity messages. Under this framework, a data-integrity message contains an entity body so that a server can declare its authorization on a message, active web intermediaries can modify the message and clients can verify it. However, it should also be backward compatible such that a normal HTTP proxy can process the non-integrity part of the response without error.

```
HTTP Status_Line
General \ Response \ Entity Headers
CRLF
(Manifest)+
(Part Headers
  Part Body)+
(Notification)*
```

Figure 1: Message Format
(where "+" denotes one or more occurrences and "*" denotes zero or more occurrences)

The format for the data-integrity message in our framework is shown in Figure 1. Their details are as follows:

- *Status Line*
  The status line in a data-integrity message is defined in the same way as that in a normal HTTP response message. The semantics of the status codes also follows those in HTTP/1.1 for status communication.

- *Headers*
  Generally speaking, the message headers are consistent with those defined in HTTP/1.1 [3]. However, some headers might lose their original meanings due to the change of the operating environment from object homogeneity to heterogeneity. As will be seen later in this section, we will analyze all the HTTP headers and propose the concept of "Part Headers" in Section 3.4 in our data-integrity language. Furthermore, we also need "DIAction", an extended HTTP response header field to indicate the intent of a data-integrity message.

- *Message Body*
  The entity body consists of one or more "manifests", one or more "parts", and zero or more "notifications". They are the important components of our language.
  Manifest: A server should provide a manifest to specify its intentions for authorizing intermediary proxies to perform value-added services on the message (See Section 3.2). A manifest might also be provided by an intermediary proxy who is authorized by the server for further task delegation.
  Part: A part is the basic unit of data content for manipulation by an intermediary. The party who provides a manifest should divide the object (or fragment of an object) into parts, each of which can be manipulated and validated separately from the rest. An intermediary proxy should modify content in the range of an authorized part, and a client might verify a message in the unit of a part. A part consists of part headers and a part body. (See Section 3.3)
  Notification: A notification is the footprint about the content modification of a part that an authorized proxy performs. Details about this concept will be discussed in Section 4.2).
  Note that the entity body of a message body might be encoded via the method specified in the "Transfer-Encoding" header field (See [3] for details).

Next, we will discuss how a server makes use of Manifest, Part and Headers to express its authorizations.

3.2. Manifest

Both a server and delegated proxies can provide manifests. The elements and the functionalities of proxies' manifests are almost the as server's one. We will cover proxies' manifests in Section 4.2 and server's manifest in this section. A manifest has two important functionalities. One is for a server to specify its authorizations. The other is to prevent its intentions from being tampered.

3.2.1. Authorization Information

We have mentioned that a server should partition its object into parts and use the part as the authorization unit. So we use a pair of tags `< PartInfo >` and `< /PartInfo >` to mark up authorizations on a part. The server should identify which part it intends to authorize via the element "`PartID`" and specify its authorizations on this part. Since the server may authorize others to do a variety of services on the part, each authorization on this part is confined via a pair of tags `< Permission >` and `< /Permission >`.

Table 1: Action, Interpretation and Roles
(n.a.: not applicable; c.o.: content owner; p.: presenter; a.o.: authorization owner)

Action	Interpretation	Possible Roles
`None`	No authorization is permitted on the part	n.a.
`Replace`	Replace content of the part with new content	c.o.
`Delete`	Cut off all the content of the part	c.o.
`Transform`	Give a new representation of the content of the part	p.
`Delegate`	Do actions or authorize others to do actions	c.o., p., a.o.

In an authorization, i.e., between < `Permission` > and < `/Permission` >, three aspects of information should be given: (i) What action(s) can be done? (ii) Who should do the action(s)? (iii) With what restriction(s) the action(s) should be done?

- `Action`:
  This element gives an authorized service. By now, our language supports four types of feasible services. The keywords "`Replace`", "`Delete`", "`Transform`" and "`Delegate`" stand for these services respectively. A keyword is also needed for the server to express a part not in demands of any services. These keywords and their corresponding meanings are listed in Table 1. If a new type of service is available, the language can be extended to support it easily.

- `Editor`:
  The element provides an authorized proxy. Here, we propose to use the URL host name of an intermediary proxy to identify it.

- `Restricts`:
  All the constraints should be declared here to confine the authorization. Usually, the constraints are related to the content's properties. For example, the server can limit the type, format, language or length of a new content provided by proxies. But for "`Delegate`" action, its meaning is much more than this. The constraints should give answers to at least these three questions. Can a delegated proxy `A` authorize a proxy `B` to do services? Can the proxy `A` (without delegation from the server) authorize the proxy `B` to do a certain service? If the answer is "yes" to the first two questions, can the proxy `B` further authorize others to do its authorized services? The answers of these questions are given by the sub-elements of the "`Restricts`" element, "`Editor`", "`Action`", and "`Depth`". (See more in Section 4.2). Note that although "`Action`" and "`Editor`" elements can have only one value, it is possible for an authorization to contain multiple of these

elements. In this case, all the specified editors will do the specified actions on a part with the same restrictions.

Two elements, "`PartDigestValue`" in a part information and "`Roles`" in an permission, have not been introduced yet. The first element is one of the protection measures against malicious intermediaries. The element "`Roles`" depicts what roles an editor might play on the data integrity framework due to their services permitted on a data integrity message. Note that for every role or service that a data-integrity intermediary does, there will be a corresponding responsibility in the data integrity framework. For example, an intermediary proxy uploading local information to a part needs to be responsible for its freshness and data validation.

Now, let us analyze what might be changed by each of the support services and conclude their possible roles in the data integrity framework below. We also list the possible roles of an action in Column 3 of Table 1.

- *Content*
  From the interpretations of "`Replace`" and "`Delete`", they modify the original content of a part. If a delegated proxy does "`Replace`" or "`Delete`" action by itself, "`Delegate`" action will also change the content of the authorized part. In these cases, an authorized proxy will play the role of `Content Owner`.

- *Representation*
  "`Transform`" action might only change the representation of an authorized part but not its content. Also, "`Delegate`" action will bring a new representation to a delegated part if a delegated proxy itself transforms the content of the part through "`Transform`" action. In these cases, an authorized proxy will play the role of `Presenter`.

- *Authorization*
  Only "`Delegate`" action might change authorizations on a part. A delegated proxy becomes an `Authorization Owner` if it authorizes others to do some services on its delegated part.

3.3. Part

A server uses < `Part` > and < `/Part` > tags to mark up a part of an object, which is defined as the basic entity for ownership and content manipulation. To decompose an object into parts, while a server might have its own rules, there are three general guidelines that we would like to suggest.

The first guideline is that each part should be independent of the other in the object. If dependency occurs between two parts, errors might occur. For example, a server asks proxies A and B to do language translation on the content of two parts a and b respectively. If there is content dependency between the two parts a and b, errors or at least inaccuracy translation might occur because separate translation might cause some of the original meanings to be lost.

The second guideline is related to the malicious proxy attack. It is advisable for a server to mark up *all* the parts of an object in lest the unmarked parts might be attacked. In this way, the integrity of the whole object can be ensured.

Lastly, the properties (or attributes) of a part need to be specified carefully. For example, the content of the object in a part is also the content of the part. < Content > and < /Content > tags are used to mark it up and "PartID" element is used to identify a part. Furthermore, it is necessary to give out properties of a part via "Headers" element.

3.4. Message and Part Headers

Under the current HTTP definition, headers are used to describe the attributes of an object. It is defined by the tag < Headers > and < /Headers >. One basic assumption behind is that the same attribute value can be applied to every single byte of the object. However, with the introduction of content heterogeneity by active web intermediaries, this assumption might not be valid to some of the headers' values. A given attribute (reflected in the header) might have different values to different parts in the same object. In this section, we would like to introduce the concept of "homogeneous" message headers and "heterogeneous" part headers for an object and define the relationship between them.

A *message header* is a HTTP header which describes a property (or attribute) of a whole web object and its value will not be affected by any intermediary's value-added services to individual parts of the object. That is, the attribute can be applied to *all* parts of an object. On the other hand, a *part header* is a HTTP header which describes a property (or attribute) of a part, defined by the tag pair < Part > and < /Part > in a web object. These headers are specified in the header line, starting with <Headers? tag and ending with <\Headers> tag. With decomposition of the object into parts for web intermediaries to work on, the attribute of interest might have different values for different parts.

On top of the current HTTP headers, there are four new headers that we introduce for a part. They are: "Content-Owner", "Presenter", "Authorization-Owner", and "URL". The first three headers record which intermediary does the services on the part and describes its role/responsibility. A data-integrity intermediary should also specify its host name in these headers if it plays the corresponding roles. The "`URL`" header is used to locate the part. These four headers will be very useful when a part is cached and it needs to be validated for reuse.

There is one intrinsic relationship between these two types of headers. Whenever an attribute of a part is described by both the message header and the part header at the same time, the latter one will override the former one. That is, the message header will lose its effect in this situation. This property is to give flexibility in the actual implementation of the system architecture and the application deployment. Note that headers specified in one part do not affect the properties of the other sibling parts.

4. Footprints Of Intermediary Proxies

One important requirement of a good data-integrity framework is for the intermediary proxies to leave footprints (or what they have done) in the message that passes through them. In the language support, the footprint will be mapped into information in three locations: part headers, notification, and possible manifests.

4.1. Data-Integrity Intermediary's Manifest

A data-integrity intermediary's manifest is an important component in our language definition for data integrity framework. Its delegated proxy's manifest plays the same role as a server's manifest. It provides authorizations to the subsequent intermediary proxies clearly and accurately. Thus it is made up of both authorization information and protection measures. Despite the similarities, however, there are two basic differences between the server's (parent's) manifest and delegated proxy's (child's) manifest.

- *Authorization Information*
  The basic units for authorization in the two manifests are different. While a delegated proxy's manifest works on one part of an object, a server's manifest works on the whole object. Thus, what the "`MessageURL`" element refers to is the URL of the authorized part, but not the object. On the other hand, although the tags " <`PartInfo`>" and "</`PartInfo`>" in the proxy's manifest mark up only some sub-part of the authorized part, the basic components of the authorization information inside is still the same. The only extra information required is the "`PartID`" element, which describes the relationship between the parent and the child manifests. A

sub-part is given an ID with a suffix ".x", where the ID stands for the parent manifest's ID and the suffix ".x" indicates which sub-part in this parent manifest is being described. In the case where the proxy does not partition its authorized part, the part ID will have a ".0" suffix.

- *Protection Measures*
  Compared to the protection measures in a server's manifest, one new key element introduced the delegated proxy's manifest is the "ParentManifestDigestValue", which specifies who authorizes the proxy to give such a manifest. On the other hand, the element "PartDigestValue" for each sub-part might be omitted. Since the proxy's manifest is generated on-on-the-fly and should be streamed from the proxy just like the server's manifest, we propose to put off the calculation of the digest value of each sub-part to the notification by the intermediary proxy. The proxy should provide both a manifest and a notification if the digest value of a sub-part is specified. However, the proxy need not give a notification if it just delegates the authorized part identified by the suffix ".0".

Note that despite the differences, information extracted from a delegated proxy's manifest is the same as, if not more than, as a server's manifest.

4.2. Notification

Notification is another key footprint element of the intermediary proxies that the data-integrity framework introduces. There are at least four considerations to construct such notification. Firstly, with the element "ManifestDigestValue", the client can find out which manifest authorizes an intermediary proxy to do the action. Secondly, the elements "Editor", "Action" and "PartID" can be used to ensure the consistency of the manifest: Who does what action on which part. Thirdly, to assure that the part received by the client is exactly what the proxy should put in the message, the proxy fills in the "PartDigiestValues" with the digest value of the part. Finally, in order to prove that the notification is really from the proxy, the proxy should sign the notification just as the server signs its manifest.

Besides the components introduced above, a notification might also include "InputDigestValue" and "PartDigestValues" elements. To assure that the authorized part received by the proxy that does the "Transform" action is not tempered by malicious intermediaries, the proxy should put the digest value of the part before transformation into the "InputDigestValue" element. The element "`PartDigestValues`" is for the intermediary proxy that does the "`Delegate`" action to record each sub-part's digest value.

5. Conclusions

In this paper, we proposed a data integrity framework with the following functionalities that a server can specify its authorizations, active web intermediaries can provide services in accordance with the server's intentions, and more importantly, a client is facilitated to verify the received message with the server's authorizations and intermediaries' traces. Our main contributions are to define a data integrity framework, its associated language specification and its associated system model to solve the data integrity problem in active, content transformation network. We also built a prototype of our data integrity model to show the practicability of our proposal.

References

1. A. Barbir, N. Mistry, R. Penno, and D. Kaplan, "A framework for OPES end to end data integrity: Virtual private content networks (VPCN)," Nov 2001. http://standards.nortelnetworks.com/opes/non-wg-doc/draft-barbir-opes-vpcn-00.txt
2. C.-H. Chi, and Y. Wu, "An XML-based data integrity service model for web intermediaries," *Proc. International Workshop on Web Content Caching and Distribution*, August 2002.
3. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L.Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt
4. J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential benefits of delta encoding and data compression for HTTP," in *Proc. SIGCOMM*, 1997, pp. 181–194.
5. J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential benefits of delta encoding and data compression for HTTP (corrected version)," Dec 1997. http://citeseer.nj.nec.com/mogul97potential.html
6. H. K. Orman, "Data integrity for mildly active content," Aug 14 2001. http://standards.nortelnetworks.com/opes/non-wg-doc/opes-data-integrityPaper.pdf
7. S. Thomas, *HTTP Essentials*. Wiley, John and Sons, 2001.