Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2012

S. Farrell
Trinity College Dublin
D. Kutscher
NEC
C. Dannewitz
University of Paderborn
B. Ohlman
Ericsson
P. Hallam-Baker
Comodo Group Inc.

October 24, 2011

The Named Information (ni) URI Scheme: Core Syntax
draft-farrell-decade-ni-00

Abstract

This document defines a URI-based name form that identifies a named object via hash-based binding. The URI name form defined is intended for use in applications that need to uniquely identify resources in a location-independent way such as accessing in-network storage (DECADE), information-centric networking and more generally. The format is designed to support a strong link to the referenced object such that the referenced object may be authenticated to the same degree as the reference to it.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 26, 2012.

Copyright Notice

Table of Contents

1. Introduction 3
2. Format 4
3. Processing NI URIs 4
3.1. Verifying URI/Resource Mappings 4
3.2. Testing for Equality 5
3.3. Mapping to HTTP(S) URLs 5
4. Examples 6
5. The Named Information URI TYPE 6
5.1. Encoding Considerations 6
5.2. Syntax 7
6. Security Considerations 7
7. Acknowledgements 8
8. IANA Considerations 8
8.1. Assignment of Network Information (ni) URI Scheme 8
8.2. Assignment of Well Known URI prefix ni 9
9. References 9
9.1. Normative References 9
9.2. Informative References 10
Authors' Addresses 10

1. Introduction

URIs [RFC3986] are used in various protocols for identifying resources. In many deployments those URIs contain strings that are hash function outputs in order to ensure uniqueness in terms of mapping the URI to a specific resource, or to make URIs hard to guess for security reasons. However, there is no standard way to interpret those strings, and so today in general only the creator of the URI knows how to use the hash function output.

For example, protocols for accessing in-network storage servers (as defined in the IETF DECADE WG) need a way to identify the stored resources uniquely and in a location-independent way so that replicas on different servers can be accessed by the same name. Also, such applications may require verifying that a resource that has been obtained actually corresponds to the name that was used to request the resource, i.e., verifying the name-content binding.

Similarly, in the context of information-centric networking [ref.netinf-design] [ref.ccn] and elsewhere there is value in being able to compare a presented resource against the URI that was de-referenced in order to access that resource. If a cryptographically-strong comparison function can be used then this allows for many forms of in-network storage, without requiring as much trust in the infrastructure used to present the resource. The outputs of hash functions can be used in this manner, if presented in a standard way.

Additional applications might include creating references from web pages delivered over HTTP/TLS; DNS resource records signed using DNSSEC or Data values embedded in certificates, CRLs, OCSP tokens and other signed data objects.

Accordingly, the "ni" URI scheme allows for checking of the integrity of the URI/resource mapping, but it is OPTIONAL for implementations to do so when sending, receiving or processing "ni" URIs.

The URI scheme defined here allows for the use of a query-string, simiilar to how query-strings are used in HTTP URLs. A companion specification [niexts] describes specific values that can be used in such query strings in for various purposese. That document also specifies additional optional algorithms for truncated hashes and for hashing of dynamic objects.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Syntax definitions in this memo are specified according to ABNF

[RFC4648].

[[Comments are included in double-square brackets, like this.]]

2. Format

In this section we provide an informal description of the ni name syntax. An ni URI consists of the following components:

Scheme Name [Required] The scheme name is 'ni'.

Colon and Slashes [Required] The literal "://"

Authority [Optional] The optional authority component may assist applications in accessing the object named by an ni URI. Note that while the ni names with and without an authority differ syntactically, both names will almost always refer to the same object.

One slash [Required] The literal "/"

Digest Algorithm [Required] The name of the digest algorithm, as specified in the IANA registry titled "Data Structure for the Security Suitability of Cryptographic Algorithms registry 'Cryptographic Algorithms'" [RFC5698].

Separator [Required] The literal ";"

Digest Value [Required] The digest value encoded in the specified encoding. The digest value MAY be trucated at a 64 byte boundary.

Query Parameter separator [Optional] '?' The query parameter separator acts a separator between the digest value and the query parameters (if specified).

Query Parameters [Optional] A tag=value list of optional query parameters as are used with HTTP URLs.

3. Processing NI URIs

3.1. Verifying URI/Resource Mappings

It is OPTIONAL for implementations to check the integrity of the URI/resource mapping when sending, receiving or processing "ni" URIs.

3.2. Testing for Equality

When verifying whether two NI URIs refer to same object, an implementation MUST only consider the Digest Algorithm identifier and the Digest Value, i.e., it MUST NOT consider the authority field or any parameters.

3.3. Mapping to HTTP(S) URLs

We define a bidirectional mapping between the ni URI scheme and a subset of the the HTTP scheme that makes use of the .well-known URI [RFC5785] by defining an "ni" suffix (see Section 8).

The HTTP(s) mapping MAY be used in any context where legacy clients without support for ni indentifiers is required without loss of interoperability or functionality. A legacy client interprets the ni identifier as an ordinary HTTP(s) URL while a ni aware client can determine the corresponding ni form of the URI and apply ni processing.

Implementations SHOULD support this mapping, in both directions. [[Not sure if we really want 2119 language for the mapping, nor if we need to specify both directions, so this is kind of a placeholder.]]

For an ni name of the form "ni://n-authority/alg;val?query-string" the corresponding HTTP URL produced by this algorithm is "http://h-authority/.well-known/ni/alg/val?query-string". If the ni name has a specified authority then the h-authority MUST have the same value. If the ni name has no authority specified (i.e. the n-authority string is empty), a h-authority value MAY be derived from the applicaiton context. For example, if the mapping is being done in the context of a web page then the origin [websec-origin] for that web site can be used. Of course, there are in general no guarantees that the object named by the ni name will be available at the corresponding HTTP URL. But in the case that any data is returned, the retreiver can determine if it is the correct content.

If an application is presented with a HTTP URL with "/.well-known/ni/" as the start of its pathname component, then the reverse mapping to an ni name either including or excluding the authority might produce an ni name that is meaningful depending on the application.

In all of the above the application MAY use the "https" URI scheme if security considerations warrant use of TLS.

4. Examples

[[Note: check examples and make sure they’re correct sometime.]]

The following digest URI specifies a reference to the text "Hello World !" using the SHA-2 algorithm with 256 bit output and no authority field:

ni:///sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc

And the same example shown with an authority would be:

ni://example.com/sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc

The following HTTP URL represents a mapping from the previous ni name based on the algorithm outlined above.

http://example.com/.well-known/ni/sha-256/
B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc

5. The Named Information URI TYPE

5.1. Encoding Considerations

[[Note that this section may change. However, the intent is that there be one and only one well defined encoding scheme for ni names. However, getting the right scheme for that, and for the URL mapping may be tricky.]]

The digest value MUST be encoded using base64url [RFC4648] encoding.

The query segment of an URI is NOT hierarchical. Thus escape encoding of slash ’/’ characters is NOT required. Since application code often attempts to enforce such encoding, decoders MUST recognize the use of URI escape encoding. Section 3.4 of [RFC3986] states that "The characters slash ("/") and question mark ("?") may represent data within the query component."

Consequently no special escaping mechanism is required for the query parameter portion of ni URIs. URI escaping is however frequently imposed automatically by scripting environments. Thus to ensure interoperability, implementations SHOULD NOT generate URIs that employ URI character escaping, and implementations MUST accept any URIs that employ URI character escaping. [[That might need to be more specific.]]

5.2. Syntax

The Named Information URI has the following syntax:

```
niname ="ni://" [ authority ] "/" alg ";" val [ "?" query ]
alg = 1*CHAR
val = 1*CHAR
```

Figure 1: ni Name syntax

The "authority" and "query" types are as in the URI specification. [RFC3986]

Implementations MUST support the sha-256 algorithm as specified in [RFC4055].

Implementations MAY support other algorithms specified in the Data Structure for the Security Suitability of Cryptographic Algorithms registry 'Cryptographic Algorithms' [RFC5698].

Note that additional algorithms are specified in the companion document to this one [niexts] that implementations can choose to support if they wish. Those algorithms use a different IANA registry defined in that document.

The "val" field MUST contain the output of applying the hash function ("alg") to its defined input, which defaults to the object bytes that are expected to be returned when the URI is de-referenced.

6. Security Considerations

No secret information is required to generate or verify an ni URI. Therefore an ni URI only provides a proof of integrity for the referenced object and the proof of integrity provided is only as good as the proof of integrity for the ni URI. In other words, the digest value can provide name-data integrity binding the ni name value to the object bytes returned when the ni name is de-referenced using some protocol.

Disclosure of an ni URI value does not necessarily entail disclosure of the referenced object but may enable an attacker to determine the contents of the referenced object by reference to a search engine or other data repository or, for highly formatted object with little variation, by simply guessing the value and checking if the digest value matches.

The integrity of the referenced content would be compromised if a

weak digest were used.

If a truncated digest is used, certain security properties MAY be affected. In general a digest algorithm is designed to produce sufficient bits to prevent a 'birthday attac' collision occuring. To ensure that the difficulty of discovering two pieces of content that result in the same digest with a work factor O(2^x) by brute force requires a digest length of 2x. Many security applications only require protection against a 2nd pre-image attack which only requires a digest length of x to achieve the same work factor.

[[Don't reduce too much, and don't rely on a digest that has been truncated as being the strength of the original digest alg.]]

7. Acknowledgements

This work has been supported by the EU FP7 project SAIL. The authors would like to thank SAIL participants to our naming discussions, especially Jean-Francois Peltier, for their input.

[[Mention folk on the WebSec list who contributed to the discussions]]

8. IANA Considerations

8.1. Assignment of Network Information (ni) URI Scheme

The procedures for registration of a URI scheme are specified in RFC 4395 [RFC4395]. The following is the proposed assignment template.

URI scheme name: ni

Status: Permanent

URI scheme syntax. See Section 5.2

URI scheme semantics. See Section 5.2

Encoding considerations. See Section 5.1

Applications/protocols that use this URI scheme name: General applicability with initial use cases provided by WEBSEC and DECADE

Interoperability considerations: TBS

Security considerations: See Section 6

Contact: TBD

Author/Change controller: IETF

References: As specified in this document

8.2. Assignment of Well Known URI prefix ni

The procedures for registration of a Well Known URI entry are specified in RFC 5785 [RFC5785]. The following is the proposed assignment template.

URI suffix: ni

Change controller: IETF

Specification document(s): This document

Related information: None

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5698] Kunz, T., Okunick, S., and U. Pordesch, "Data Structure for the Security Suitability of Cryptographic Algorithms (DSSC)", RFC 5698, November 2009.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

9.2. Informative References

[niexts] Hallam-Baker, P., Stradling, R., Farrell, S., Kutscher, C., and B. Ohlman, "The Network Information (ni) URI Scheme: Parameters", draft-hallambaker-decade-ni-params-00 (work in progress), October 2011.

[ref.ccn] Jacobsen, K, D, F, H, and L, "Networking Named Content", CoNEXT 2009 , December 2009.

[ref.netinf-design] Ahlgren, D'Ambrosio, Dannewitz, Marchisio, Marsh, Ohlman, Pentikousis, Rembarz, Strandberg, and Vercellone, "Design Considerations for a Network of Information", Re-Arch 2008 Workshop , December 2008.

[websec-origin] Barth, A., "The Web Origin Concept", draft-ietf-websec-origin-06 (work in progress), October 2011.

Authors' Addresses

Stephen Farrell
Trinity College Dublin
Dublin, 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie

Dirk Kutscher
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: kutscher@neclab.eu

Christian Dannewitz
University of Paderborn
Paderborn
Germany

Email: cdannewitz@upb.de

Borje Ohlman
Ericsson
Stockholm S-16480
Sweden

Email: Borje.Ohlman@ericsson.com

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: February 4, 2013

S. Farrell
Trinity College Dublin
D. Kutscher
NEC
C. Dannewitz
University of Paderborn
B. Ohlman
A. Keranen
Ericsson
P. Hallam-Baker
Comodo Group Inc.

August 3, 2012

Naming Things with Hashes
draft-farrell-decade-ni-10

Abstract

This document defines a set of ways to identify a thing (a digital object in this case) using the output from a hash function, specifying a new URI scheme for this, a way to map those to http URLs, and binary and human "speakable" formats for these names. The various formats are designed to support, but not require, a strong link to the referenced object such that the referenced object may be authenticated to the same degree as the reference to it. This work is motivated as a way to standardise current uses of hash outputs in URLs and to support new information-centric applications and other uses of hash outputs in protocols.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 4, 2013.

Copyright Notice

Table of Contents

1. Introduction 3
2. Hashes are what Count 4
3. Named Information (ni) URI Format 6
3.1. Content Type Query String Attribute 8
4. .well-known URI 9
5. URL Segment Format 10
6. Binary Format 10
7. Human-speakable (nih) URI Format 11
8. Examples 12
8.1. Hello World! 13
8.2. Public Key Examples 13
8.3. nih Usage Example 14
9. IANA Considerations 15
9.1. Assignment of ni URI Scheme 15
9.2. Assignment of nih URI Scheme 15
9.3. Assignment of .well-known 'ni' URI 16
9.4. Creation of Named Information Hash Algorithm Registry . . 16
9.5. Creation of Named Information Parameter Registry 17
10. Security Considerations 18
11. Acknowledgments 20
12. References 20
12.1. Normative References 20
12.2. Informative References 21
Authors' Addresses 22

1. Introduction

Identifiers -- names or locators -- are used in various protocols to identify resources. In many scenarios, those identifiers contain values that are obtained from hash functions. Different deployments have chosen different ways to include the hash function outputs in their identifiers, resulting in interoperability problems.

This document defines a "Named Information" identifier, which provides a set of standard ways to use hash function outputs in names. We begin with a few example uses for various ways to include hash function output in a name, with the specifics defined later in this document. Figure 1 shows an example of the Named Information (ni) URI scheme that this document defines.

```
ni:///sha-256;UyaQV-Ev4rdLoHyJJWCi11OHfrYv9E1aGQAlMO2X_-Q
```

Figure 1: Example ni URI

Hash function outputs can be used to ensure uniqueness in terms of mapping URIs [RFC3986] to a specific resource, or to make URIs hard to guess for security reasons. Since there is no standard way to interpret those strings today, in general only the creator of the URI knows how to use the hash function output. Other protocols, such as application layer protocols for accessing "smart objects" in constrained environments also require more compact (e.g., binary) forms of such identifiers. In yet other situations people may have to speak such values, e.g., in a voice call, (see Section 8.3), in order to confirm the presence or absence of a resource.

As another example, protocols for accessing in-network storage servers need a way to identify stored resources uniquely and in a location-independent way so that replicas on different servers can be accessed by the same name. Also, such applications may require verification that a resource representation that has been obtained actually corresponds to the name that was used to request the resource, i.e., verifying the binding between the data and the name, which is here termed name-data integrity.

Similarly, in the context of information-centric networking [ref.netinf-design] [ref.ccn] and elsewhere there is value in being able to compare a presented resource against the URI that was used to access that resource. If a cryptographically-strong comparison function can be used then this allows for many forms of in-network storage, without requiring as much trust in the infrastructure used to present the resource. The outputs of hash functions can be used in this manner, if thry are presented in a standard way.

Additional applications might include creating references from web pages delivered over HTTP/TLS; DNS resource records signed using DNSSEC or data values embedded in certificates, Certificate Revocation Lists (CRLs), or other signed data objects.

The Named Identifier can be represented in a number of ways: using the "ni" URI scheme that we specifically define for the name (which is very similar to the "magnet link" that is informally defined in other protocols [magnet]), or using other mechanisms also defined herein. However it is represented, the Named Identifier *names* a resource, and the mechanism used to dereference the name and to *locate* the named resource needs to be known by the entity that dereferences it.

Media content-type, alternative locations for retrieval and other additional information about a resource named using this scheme can be provided using a query string. A companion specification [I-D.hallambaker-decade-ni-params] describes specific values that can be used in such query strings for these various purposes and other extensions to this basic format specification.

In addition, we also define a ".well-known" URL equivalent, and a way to include a hash as a segment of an HTTP URL, as well as a binary format for use in protocols that require more compact names and a human-speakable text form that could be used, e.g., for reading out (parts of) the name over a voice connection.

Not all uses of these names require use of the full hash output - truncated hashes can be safely used in some environments. For this reason, we define a new IANA registry for hash functions to be used with this specification so as not to mix strong and weak (truncated) hash algorithms in other protocol registries.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Syntax definitions in this memo are specified according to ABNF [RFC5234].

2. Hashes are what Count

This section contains basic considerations related to how we use hash function outputs that are common to all formats.

When comparing two names of the form defined here, an implementation MUST only consider the digest algorithm and the digest value, i.e.,

it MUST NOT consider other fields defined below (such as an authority field from a URI or any parameters). Implementations MUST consider two hashes identical, regardless of encoding, if the decoded hashes are based on the same algorithm and have the same length and the same binary value. In that case, the two names can be treated as referring to the same thing.

The sha-256 algorithm as specified in [SHA-256] is mandatory to implement, that is, implementations MUST be able to generate/send and to accept/process names based on a sha-256 hash. However implementations MAY support additional hash algorithms and MAY use those for specific names, for example in a constrained environment where sha-256 is non-optimal or where truncated names are needed to fit into corresponding protocols (when a higher collision probability can be tolerated).

Truncated hashes MAY be supported. When a hash value is truncated the name MUST indicate this. Therefore we use different hash algorithm strings for these, such as sha-256-32 for a 32-bit truncation of a sha-256 output. A 32-bit truncated hash is essentially useless for security in almost all cases, but might be useful for naming. With current best practices [RFC3766] very few, if any, applications making use of names with less than 100 bit long hashes will have useful security properties.

When a hash value is truncated to N bits the left-most N bits, that is, the most significant N bits in network byte order, from the binary representation of the hash value MUST be used as the truncated value. An example of a 128-bit hash output truncated to 32 bits is shown in Figure 2.

```
                  128-bit hash: 0x265357902fe1b7e2a04b897c6025d7a2
          32-bit truncated hash: 0x26535790
```

Figure 2: Example of Truncated Hash

When the input to the hash algorithm is a public key value, as may be used by various security protocols, the hash SHOULD be calculated over the public key in an X.509 SubjectPublicKeyInfo structure (Section 4.1 of [RFC5280]). This input has been chosen primarily for compatibility with DANE [I-D.ietf-dane-protocol], but also includes any relevant public key parameters in the hash input, which is sometimes necessary for security reasons. This does not force use of X.509 or full compliance with [RFC5280] since formatting any public key as a SubjectPublicKeyInfo is relatively straightforward and well supported by libraries.

Any of the formats defined below can be used to represent the resulting name for a public key.

Other than in the above special case where public keys are used, we do not specify the hash function input here. Other specifications are expected to define this.

3. Named Information (ni) URI Format

A Named Information (ni) URI consists of the following nine components:

Scheme Name The scheme name is 'ni'.

Colon and Slashes The literal "://"

Authority The optional authority component may assist applications in accessing the object named by an ni URI. There is no default value for the authority field. (See [RFC3986] Section 3.2.2 for details.) While ni names with and without an authority differ syntactically from ni names with different authorities, all three refer to the same object if and only if the digest algorithm, length, and value are the same.

One slash The literal "/"

Digest Algorithm The name of the digest algorithm, as specified in the IANA registry defined in Section 9.4 below.

Separator The literal ";"

Digest Value The digest value MUST be encoded using the base64url [RFC4648] encoding, with no "=" padding characters.

Query Parameter separator '?' The query parameter separator acts as a separator between the digest value and the query parameters (if specified). For compatibility with IRIs, non-ASCII characters in the query part MUST be encoded as UTF-8, and the resulting octets MUST be %-encoded (see [RFC3986] Section 2.1).

Query Parameters A tag=value list of optional query parameters as are used with HTTP URLs [RFC2616] with a separator character '&' between each. For example, "foo=bar&baz=bat"

It is OPTIONAL for implementations to check the integrity of the URI/resource mapping when sending, receiving or processing "ni" URIs.

Escaping of characters follows the rules in RFC 3986. This means that %-encoding is used to distinguish between reserved and unreserved functions of the same character in the same URI component. As an example, an ampersand ('&') is used in the query part to separate attribute-value pairs; an ampersand in a value therefore has to be escaped as '%26'. Note that the set of reserved characters differs for each component, as an example, a slash ('/') does not have any reserved function in a query part and therefore does not have to be escaped. However, it can still appear escaped as '%2f' or '%2F', and implementations have to be able to understand such escaped forms. Also note that any characters outside those allowed in the respective URI component have to be escaped.

The Named Information URI adapts the URI definition from the URI Generic Syntax [RFC3986]. We start with the base URI production:

```
URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
    ; from RFC 3986
```

Figure 3: URI syntax

Adapting that for the Named Information URI:

```
NI-URI         = ni-scheme ":" ni-hier-part [ "?" query ]
    ; adapted from "URI" in RFC 3986
    ; query is from RFC 3986, Section 3.4
ni-scheme      = "ni"
ni-hier-part   = "//" [ authority ] "/" alg-val
    ; authority is from RFC 3986, Section 3.2
alg-val        = alg ";" val
    ; adapted from "hier-part" in RFC 3986
alg            = 1*unreserved
val            = 1*unreserved
    ; unreserved is from RFC 3986, Section 2.3
```

Figure 4: ni Name syntax

The "val" field MUST contain the output of base64url encoding (with no "=" padding characters) the result of applying the hash function ("alg") to its defined input, which defaults to the object bytes that are expected to be returned when the URI is dereferenced.

Relative ni URIs can occur. In such cases, the algorithm in [RFC3986] Section 5 applies. As an example, in Figure 5, the absolute URI for "this third document" is "ni://example.com/sha-256-128;...".

```
<html>
 <head>
   <title>ni: relative URI test</title>
   <base href="ni://example.com">
 </head>

 <body>
   <p>Please check <a href="sha-256;f4OxZX...">this document</a>.
     and <a href="sha-256;UyaQV...">this other document</a>.
     and <a href="sha-256-128;...">this third document</a>.
   </p>
 </body>
</html>
```

Figure 5: Example HTML with relative ni URI

The authority field in an ni URI is not quite the same as that from an HTTP URL, even though the same values (e.g., DNS names) may be usefully used in both. For an ni URI, the authority does not control nearly as much of the structure of the "right hand side" of the URI. With ni URIs we also define standard query string attributes and of cousrse have a strictly defined way to include the hash value.

Internationalisation of strings within ni names is handled exactly as for http URIs - see [I-D.ietf-httpbis-p1-messaging] Section 2.7.

3.1. Content Type Query String Attribute

The semantics of a digest being used to establish a secure reference from an authenticated source to an external source may be a function of associated meta data such as the content type. The Content Type "ct" parameter specifies the MIME Content Type of the associated data as defined in [I-D.ietf-appsawg-media-type-regs]. See Section 9.5 for the associated IANA registry for ni parameter names. as shown in Figure 6. Implementations of this specification MUST support parsing the ct= query string attribute name.

```
ni:///sha-256-32;f4OxZQ?ct=text/plain
```

Figure 6: Example ni URI with Content Type

Protocols making use of ni URIs will need to specify how to verify name-data integrity for the MIME Content Types that they need to process and will need to take into account possible Content-Transfer-Encodings and other aspects of MIME encoding.

Implementations of this specification SHOULD support name-data

integrity validation for at least the application/octet-stream Content Type with no explicit Content-Transfer-Encoding (which is equivalent to binary). Additional Content Types and Content-Transfer- Encodings can of course also be supported, but are OPTIONAL. Note that the hash is calculated after the Content Transfer Encoding is removed, so it is applied to the raw data.

If a) the user agent is sensitive to the Content Type and b) the ni name used has a ct= query string attribute and c) the object is retrieved (from a server) using a protocol that specifies a Content Type, then, if the two Content Types match, all is well. If, in this situation, the Content Types do not match, then the client SHOULD handle that situation as a potential security error. Content Type matching rules are defined in [RFC2045] Section 5.1.

4. .well-known URI

We define a mapping between URIs following the ni URI scheme and HTTP [RFC2616] or HTTPS [RFC2818] URLs that makes use of the .well-known URI [RFC5785] by defining an "ni" suffix (see Section 9).

The HTTP(S) mapping MAY be used in any context where clients with support for ni URIs are not available.

Since the .well-known name-space is not intended for general information retrieval, if an application de-references a .well-known/ni URL via HTTP(S), then it will often receive a 3xx HTTP re-direction response. A server responding to a request for a .well-known/ni URL will often therefore return a 3xx response and a client sending such a request MUST be able to handle that, as should any fully compliant HTTP [RFC2616] client.

For an ni name of the form "ni://n-authority/alg;val?query-string" the corresponding HTTP(S) URL produced by this mapping is "http://h-authority/.well-known/ni/alg/val?query-string", where "h-authority" is derived as follows: If the ni name has a specified authority (i.e., the n-authority is non-empty) then the h-authority MUST have the same value. If the ni name has no authority specified (i.e., the n-authority string is empty), a h-authority value MAY be derived from the application context. For example, if the mapping is being done in the context of a web page then the origin [RFC6454] for that web site can be used. Of course, there are in general no guarantees that the object named by the ni URI will be available via the corresponding HTTP(S) URL. But in the case that any data is returned, the retriever can determine whether or not it is content that matches the ni URI.

If an application is presented with a HTTP(S) URL with "/.well-known/ni/" as the start of its pathname component, then the reverse mapping to an ni URI either including or excluding the authority might produce an ni URI that is meaningful, but there is no guarantee that this will be the case.

When mapping from an ni URI to a .well-known URL, an implementation will have to decide between choosing an "http" or "https" URL. If the object referenced does in fact match the hash in the URL, then there is arguably no need for additional data integrity, if the ni URI or .well-known URL was received "securely." However TLS also provides confidentiality, so there can still be reasons to use the "https" URL scheme even in this case. Additionally, web server policy such as [I-D.ietf-websec-strict-transport-sec] may dictate that data might only be available over "https". In general however, whether to use "http" or "https" is something that needs to be decided by the application.

5. URL Segment Format

Some applications may benefit from using hashes in existing HTTP URLs or other URLs. To do this one simply uses the "alg-val" production from the ni name scheme ABNF which may be included for example in the pathname, query string or even fragment components of HTTP URLs [RFC2616]. In such cases there is nothing present in the URL that ensures that a client can depend on compliance with this specification, so clients MUST NOT assume that any URL with a pathname component that matches the "alg-val" production was in fact produced as a result of this specification. That URL might or might not be related to this specification, only the context will tell.

6. Binary Format

If a more space-efficient version of the name is needed, the following binary format can be used. The binary format name consists of two fields: a header and the hash value. The header field defines how the identifier has been created and the hash value contains a (possibly truncated) result of a one-way hash over whatever is being identified by the hash value. The binary format of a name is shown in Figure 7.

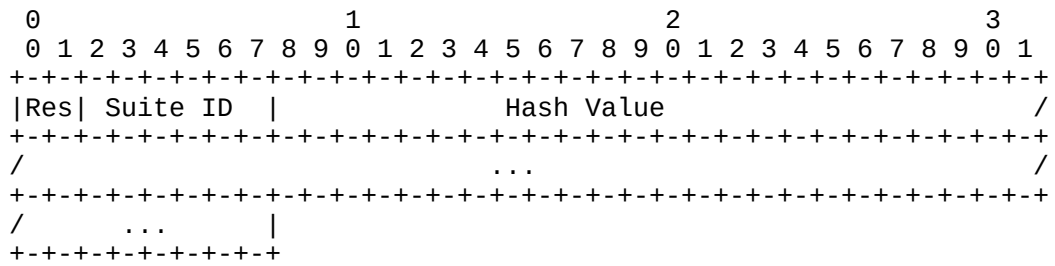


Figure 7: Binary Name Format

The Res field is a reserved 2-bit field for future use and MUST be set to zero for this specification and ignored on receipt.

The hash algorithm and truncation length are specified by the Suite ID. For maintaining efficient encoding for the binary format, only a few hash algorithms and truncation lengths are supported. See Section 9.4 for details.

A hash value that is truncated to 120 bits will result in the overall name being a 128-bit value which may be useful for protocols that can easily use 128-bit identifiers.

7. Human-speakable (nih) URI Format

Sometimes a resource may need to be referred to via a name in a format that is easy for humans to read out, and less likely to be ambiguous when heard. This is intended to be usable for example over the phone in order to confirm the (current or future) presence or absence of a resource. This "confirmation" use-case described further in Section 8.3 is the main current use-case for nih URIs.

The ni URI format is not well-suited for this, as, for example, base64url uses both upper and lower case which can easily cause confusion. For this particular purpose, ("speaking" the value of a hash output) the more verbose but less ambiguous (when spoken) nih URI scheme is defined. "nih" stands for "Named Information for Humans." (Or possibly "Not Invented Here," which is clearly false, and therefore worth including [RFC5513]:-)

The justification for nih being a URI scheme is that that can help a user agent for the speaker to better display the value, or help a machine to better speak or recognise the value when spoken. We do not include the query string since there is no way to ensure that its value might be spoken unambiguously, and similarly for the authority, where e.g., some internationalised forms of domain name might not be

easy to speak and comprehend easily. This leaves the hash value as the only part of the ni URI that we feel can be usefully included. But since speakers or listeners (or speech recognition) may err, we also include a check-digit to catch common errors, and allow for the inclusion of "-" separators to make nih URIs more easy to read out.

Fields in nih URIs are separated by a semi-colon (;) character. The first field is a hash algorithm string, as in the ni URI format. The hash value is represented using lower-case ASCII hex characters, for example an octet with the decimal value 58 (0x3A) is encoded as '3a'. This is the same as base16 encoding as defined in RFC 4648 [RFC4648] except using lower-case letters. Separators ("-" characters) MAY be interspersed in the hash value in any way to make those easier to read, typically grouping four or six characters with a separator between.

The hash value MAY be followed by a semi-colon ';' then a checkdigit. The checkdigit MUST be calculated using Luhn's mod N algorithm (with N=16) as defined in [ISOIEC7812], (see also http://en.wikipedia.org/wiki/Luhn_mod_N_algorithm). The input to the calculation is the ASCII-HEX encoded hash value (i.e., "sepval" in the ABNF production below) but with all "-" separator characters first stripped out. This maps the ASCII-HEX so that '0'=0,...'9'=9,'a'=10,...'f'=15. None of the other fields, nor any "-" separators, are input when calculating the checkdigit.

```
humanname  = "nih:" alg-sepval [ ";" checkdigit ]
alg-sepval = alg ";" sepval
sepval     = 1*(ahlc / "-")
ahlc       =  DIGIT / "a" / "b" / "c" / "d" / "e" / "f"
      ; DIGIT is defined in RFC 5234 and is 0-9
checkdigit = ahlc
```

Figure 8: Human-speakable syntax

For algorithms that have a Suite ID reserved (see Figure 11), the alg field MAY contain the ID value as a ASCII encoded decimal number instead of the hash name string (for example, "3" instead of "sha-256-120"). Implementations MUST be able to match the decimal ID values for the algorithms and hash lengths that they support even if they do not support the binary format.

There is no such thing as a relative nih URI.

8. Examples

8.1. Hello World!

The following ni URI is generated from the text "Hello World!" (without the quotes, being 12 characters), using the sha-256 algorithm shown with and without an authority field:

ni:///sha-256;f4OxZX_x_FO5LcGBSKHWXfwtSx-j1ncoSt3SABJtkGk

ni://example.com/sha-256;f4OxZX_x_FO5LcGBSKHWXfwtSx-j1ncoSt3SABJtkGk

The following HTTP URL represents a mapping from the previous ni name based on the algorithm outlined above.

http://example.com/.well-known/ni/sha-256/
f4OxZX_x_FO5LcGBSKHWXfwtSx-j1ncoSt3SABJtkGk

8.2. Public Key Examples

Given the DER-encoded SubjectPublicKeyInfo in Figure 9 we derive the names shown in Figure 10 for this value.

```
0000000 30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01
0000020 01 05 00 03 82 01 0f 00 30 82 01 0a 02 82 01 01
0000040 00 a2 5f 83 da 9b d9 f1 7a 3a 36 67 ba fd 5a 94
0000060 0e cf 16 d5 5a 55 3a 5e d4 03 b1 65 8e 6d cf a3
0000100 b7 db a4 e7 cc 0f 52 c6 7d 35 1d c4 68 c2 bd 7b
0000120 9d db e4 0a d7 10 cd f9 53 20 ee 0d d7 56 6e 5b
0000140 7a ae 2c 5f 83 0a 19 3c 72 58 96 d6 86 e8 0e e6
0000160 94 eb 5c f2 90 3e f3 a8 8a 88 56 b6 cd 36 38 76
0000200 22 97 b1 6b 3c 9c 07 f3 4f 97 08 a1 bc 29 38 9b
0000220 81 06 2b 74 60 38 7a 93 2f 39 be 12 34 09 6e 0b
0000240 57 10 b7 a3 7b f2 c6 ee d6 c1 e5 ec ae c5 9c 83
0000260 14 f4 6b 58 e2 de f2 ff c9 77 07 e3 f3 4c 97 cf
0000300 1a 28 9e 38 a1 b3 93 41 75 a1 a4 76 3f 4d 78 d7
0000320 44 d6 1a e3 ce e2 5d c5 78 4c b5 31 22 2e c7 4b
0000340 8c 6f 56 78 5c a1 c4 c0 1d ca e5 b9 44 d7 e9 90
0000360 9c bc ee b0 a2 b1 dc da 6d a0 0f f6 ad 1e 2c 12
0000400 a2 a7 66 60 3e 36 d4 91 41 c2 f2 e7 69 39 2c 9d
0000420 d2 df b5 a3 44 95 48 7c 87 64 89 dd bf 05 01 ee
0000440 dd 02 03 01 00 01

0000000 53 26 90 57 e1 2f e2 b7 4b a0 7c 89 25 60 a2 d7
0000020 53 87 7e b6 2f f4 4d 5a 19 00 25 30 ed 97 ff e4
```

Figure 9: A SubjectPublicKeyInfo used in examples and its sha-256 hash

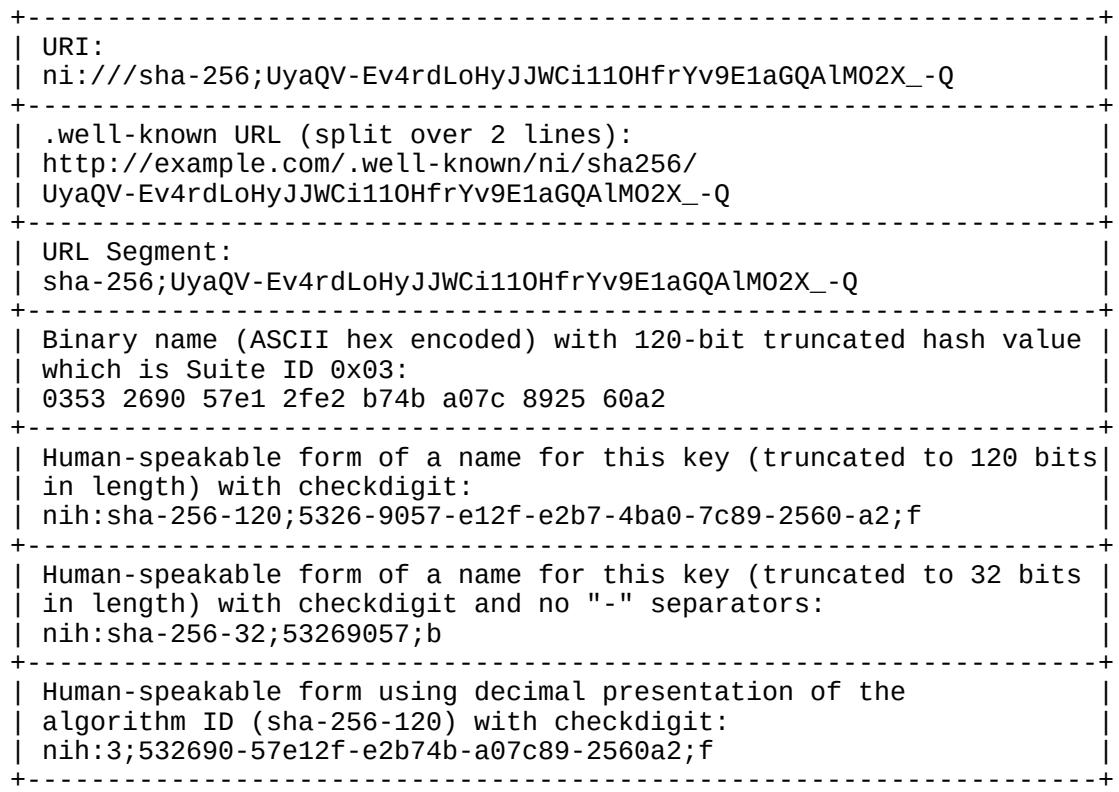


Figure 10: Example Names

8.3. nih Usage Example

Alice has set up a server node with an RSA key pair. She uses an ni URI as the name for the public key that corresponds to the private key on that box. Alice's node might identify itself using that ni URI in some protocol.

Bob would like to believe that its really Alice's node when his node interacts with the network and asks his friend Alice to tell him what public key she uses. Alice hits the "tell someone the name of the public key" button on her admin user interface, and that displays the nih URI and says "tell this to your buddy." She phones Bob and reads the nih URI to him.

Bob types that in to his "manage known nodes" admin application, (or lets that application listen to part of the call), which can regenerate the ni URI and store that or some equivalent. Then when Bob's node interacts with Alice's node it can more safely accept a

signature or encrypt data to Alice's node.

9. IANA Considerations

9.1. Assignment of ni URI Scheme

The procedures for registration of a URI scheme are specified in RFC 4395 [RFC4395]. The following is the proposed assignment template.

URI scheme name: ni

Status: Permanent

URI scheme syntax. See Section 3

URI scheme semantics. See Section 3

Encoding considerations. See Section 3

Applications/protocols that use this URI scheme name: General applicability.

Interoperability considerations: Defined here.

Security considerations: See Section 10

Contact: Stephen Farrell, stephen.farrell@cs.tcd.ie

Author/Change controller: IETF

References: As specified in this document

9.2. Assignment of nih URI Scheme

The procedures for registration of a URI scheme are specified in RFC 4395 [RFC4395]. The following is the proposed assignment template.

URI scheme name: nih

Status: Permanent

URI scheme syntax. See Section 7

URI scheme semantics. See Section 7

Encoding considerations. See Section 7

Applications/protocols that use this URI scheme name: General applicability.

Interoperability considerations: Defined here.

Security considerations: See Section 10

Contact: Stephen Farrell, stephen.farrell@cs.tcd.ie

Author/Change controller: IETF

References: As specified in this document

9.3. Assignment of .well-known 'ni' URI

The procedures for registration of a Well Known URI entry are specified in RFC 5785 [RFC5785]. The following is the proposed assignment template.

URI suffix: ni

Change controller: IETF

Specification document(s): This document

Related information: None

9.4. Creation of Named Information Hash Algorithm Registry

IANA is requested to create a new registry for hash algorithms as used in the name formats specified here and called the "Named Information Hash Algorithm Registry". Future assignments are to be made through Expert Review [RFC5226]. This registry has five fields, the suite ID, the hash algorithm name string, the truncation length, the underlying algorithm reference and a status field that indicates if algorithm is current or deprecated and should no longer be used. The status field can have the value "current" or "deprecated". Other values are reserved for possible future definition.

If the status is "current", then that does not necessarily mean that the algorithm is "good" for any particular purpose, since the cryptographic strength requirements will be set by other applications or protocols.

A request to mark an entry as "deprecated" can be done by sending a mail to the Designated Expert. Before approving the request, the community MUST be consulted via a "call for comments" of at least two weeks by sending a mail to the IETF discussion list.

Initial values are specified below. The Designated Expert SHOULD generally approve additions that reference hash algorithms that are widely used in other IETF protocols. In addition, the Designated Expert SHOULD NOT accept additions where the underlying hash function (with no truncation) is considered weak for collisions. Part of the reasoning behind this last point is that inclusion of code for weak hash functions, e.g. the MD5 algorithm, can trigger costly false-positives if code is audited for inclusion of obsolete ciphers. See for example [RFC6149],[RFC6150] and [RFC6151] for some hash functions that are considered obsolete in this sense.

The suite ID field ("ID") can be empty, or can have values between 0 and 63, inclusive. Because there are only 64 possible values, this field is OPTIONAL (leaving it empty if omitted). Where the binary format is not expected to be used for a given hash algorithm, this field SHOULD be omitted. If an entry is registered without a suite ID, the Designated Expert MAY allow for later allocation of a suite ID, if that appears warranted. The Designated Expert MAY consult the community via a "call for comments" by sending a mail to the IETF discussion list before allocating a suite ID.

ID	Hash name string	Value length	Reference	Status
0	Reserved			
1	sha-256	256 bits	[SHA-256]	current
2	sha-256-128	128 bits	[SHA-256]	current
3	sha-256-120	120 bits	[SHA-256]	current
4	sha-256-96	96 bits	[SHA-256]	current
5	sha-256-64	64 bits	[SHA-256]	current
6	sha-256-32	32 bits	[SHA-256]	current
32	Reserved			

Figure 11: Suite Identifiers

The Suite ID value 32 is reserved for compatibility with ORCHIDs [RFC4843].

The referenced hash algorithm matching to the Suite ID, truncated to the length indicated, according to the description given in Section 2, is used for generating the hash. The Designated Expert is responsible for ensuring that the document referenced for the hash algorithm meets the "specification required" rule."

9.5. Creation of Named Information Parameter Registry

IANA is requested to create a new registry entitled "Named Information URI Parameter Definitions".

The policy for future assignments to the registry is Expert Review, and as for the ni Hash Algorithm Registry above, the Designated Expert is responsible for ensuring that the document referenced for the paramater definition meets the "specification required" rule."

The fields in this registry are the parameter name, a description and a reference. The parameter name MUST be such that it is suitable for use as a query string parameter name in an ni URI. (See Section 3.)

The initial contents of the registry are:

Parameter	Meaning	Reference
ct	Content Type	[RFC-THIS]

10. Security Considerations

No secret information is required to generate or verify a name of the form described here. Therefore a name like this can only provide evidence for the integrity for the referenced object and the proof of integrity provided is only as good as the proof of integrity for the name from which we started. In other words, the hash value can provide a name-data integrity binding between the name and the bytes returned when the name is de-referenced using some protocol.

Disclosure of a name value does not necessarily entail disclosure of the referenced object but may enable an attacker to determine the contents of the referenced object by reference to a search engine or other data repository or, for a highly formatted object with little variation, by simply guessing the value and checking if the digest value matches. So the fact that these names contain hashes does not protect the confidentiality of the object that was input to the hash.

The integrity of the referenced content would be compromised if a weak hash function were used. SHA-256 is currently our preferred hash algorithm which is why we’ve only added SHA-256 based suites to the initial IANA registry.

If a truncated hash value is used, certain security properties will be affected. In general a hash algorithm is designed to produce sufficient bits to prevent a ’birthday attack’ collision occurring. To ensure that the difficulty of discovering two pieces of content that result in the same digest with a work factor O(2^x) by brute force requires a digest length of 2x. Many security applications only require protection against a 2nd pre-image attack which only requires a digest length of x to achieve the same work factor.

Basically, the shorter the hash value used, the less security benefit you can possibly get.

An important thing to keep in mind is not to make the mistake of thinking two names are the same when they aren't. For example, a name with a 32 bit truncated sha-256 hash is not the same as a name with the full 256 bits of hash output, even if the hash value for one is a prefix of that for the other.

The reason for this is that if an application treats those as the same name then that might open up a number of attacks. For example, if I publish an object with the full hash, then I probably (in general) don't want some other application to treat a name with just the first 32 bits of that as referring to the same thing, since the 32 bit name will have lots of colliding objects. If ni or nih URIs become widely used, there will be many cases where names will occur more than once in application protocols, and it'll be unpredictable which instance of the name would be used for name-data integrity checking, leading to threats. For this reason, we require that the algorithm, length and value all match before we consider two names to be the same.

The fact that an ni URI includes a domain name in the authority field by itself implies nothing about the relationship between the owner of the domain name and any content referenced by that URI. While a name-data integrity service can be provided using ni URIs, that does not in any sense validate the authority part of the name. For example, there is nothing to stop anyone creating an ni URI containing a hash of someone else's content. Application developers MUST NOT assume any relationship between the registrant of the domain name that is part of an ni URI and some matching content just because the ni URI matches that content.

If name-data integrity is successfully validated, and the hash is strong and long enough, then the "web origin" [RFC6454] for the bytes of the named object is really going to be the place from which you got the ni name and not the place from which you got the bytes of the object. This appears to offer a potential benefit if using ni names for, for example, scripts included from a HTML page accessed via server-authenticated https. If name-data integrity is not validated (and it is optional), or fails, then the web origin is, as usual, the place from which the object bytes were received. Applications making use of ni names SHOULD take this into account in their trust models.

Some implementations might mis-handle ni URIs that include non-base64 characters, whitespace or other non-conforming strings and that could lead to erroneously considering names to be the same when they are not. An ni URI that is malformed in such ways MUST NOT be treated as

matching any other ni URI. Implementers need to check the behaviour of libraries for such parsing problems.

11. Acknowledgments

This work has been supported by the EU FP7 project SAIL. The authors would like to thank SAIL participants to our naming discussions, especially Jean-Francois Peltier, for their input.

The authors would also like to thank Carsten Bormann, Martin Durst, Tobias Heer, Bjoern Hoehrmann, Tero Kivinen, Barry Leiba, Larry Masinter, David McGrew, Alexey Melnikov, Bob Moskowitz, Jonathan Rees, Eric Rescorla, Zach Shelby, Martin Thomas, for their comments and input to the document. Thanks, in particular, to James Manger for correcting the examples.

12. References

12.1. Normative References

[I-D.ietf-appsawg-media-type-regs]
Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", draft-ietf-appsawg-media-type-regs-14 (work in progress), June 2012.

[I-D.ietf-httpbis-p1-messaging]
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part 1: Message Routing and Syntax"", draft-ietf-httpbis-p1-messaging-20 (work in progress), July 2012.

[ISOIEC7812]
ISO, ""ISO/IEC 7812-1:2006 Identification cards -- Identification of issuers -- Part 1: Numbering system",", October 2006, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39698>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,

Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[SHA-256] NIST, "United States National Institute of Standards and Technology (NIST), FIPS 180-3: Secure Hash Standard", October 2008, <http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf>.

12.2. Informative References

[I-D.hallambaker-decade-ni-params]
Hallam-Baker, P., Stradling, R., Farrell, S., Kutscher, D., and B. Ohlman, "The Named Information (ni) URI Scheme: Optional Features", draft-hallambaker-decade-ni-params-03 (work in progress), June 2012.

[I-D.ietf-dane-protocol]
Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", draft-ietf-dane-protocol-23 (work in progress), June 2012.

[I-D.ietf-websec-strict-transport-sec]

Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", draft-ietf-websec-strict-transport-sec-11 (work in progress), July 2012.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, April 2004.

[RFC4843] Nikander, P., Laganier, J., and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)", RFC 4843, April 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5513] Farrel, A., "IANA Considerations for Three Letter Acronyms", RFC 5513, April 1 2009.

[RFC6149] Turner, S. and L. Chen, "MD2 to Historic Status", RFC 6149, March 2011.

[RFC6150] Turner, S. and L. Chen, "MD4 to Historic Status", RFC 6150, March 2011.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[magnet] Wikipedia article, "Magnet URI Scheme", April 2012, <http://en.wikipedia.org/wiki/Magnet_link>.

[ref.ccn] Jacobson at al., "Networking Named Content", CoNEXT 2009 , December 2009.

[ref.netinf-design] Ahlgren, D’Ambrosio, Dannewitz, Marchisio, Marsh, Ohlman, Pentikousis, Rembarz, Strandberg, and Vercellone, "Design Considerations for a Network of Information", Re-Arch 2008 Workshop , December 2008.

Authors' Addresses

Stephen Farrell
Trinity College Dublin
Dublin, 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie

Dirk Kutscher
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: kutscher@neclab.eu

Christian Dannewitz
University of Paderborn
Paderborn
Germany

Email: cdannewitz@upb.de

Borje Ohlman
Ericsson
Stockholm S-16480
Sweden

Email: Borje.Ohlman@ericsson.com

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2012

P. Hallam-Baker
Comodo Group Inc.
R. Stradling
Comodo CA Ltd.
S. Farrell
Trinity College Dublin
D. Kutscher
NEC
B. Ohlman
Ericsson

October 24, 2011

The Named Information (ni) URI Scheme: Parameters
draft-hallambaker-decade-ni-params-00

Abstract

This document specifies some optional algorithms and parameters that may be used in the query string of ni URIs.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 26, 2012.

Copyright Notice

Table of Contents

1. Introduction 3
2. Additional Algorithms 3
2.1. Truncated Hashes 3
2.2. Hashed Dynamic Content 4
3. Query String Paramaeters 5
3.1. Digest with Content Type 5
3.2. Additional Locators 5
3.3. Digest with Decryption Key 6
4. Security Considerations 7
5. IANA Considerations 7
5.1. Creation of ni additional algorithms registry 7
5.2. Creation of ni parameter registry 8
6. Normative References 8
Authors' Addresses 9

1. Introduction

The ni URI scheme [nischeme] supports extensibility in terms of the algorithm used to derive a value (normally, but not always a strong digest algorithm) and via support for a query-string thay can contain a list of key=value pairs. This document defines some uses for both of these extensibility points and creates IANA registries that can be used to register additional algorithms and key strings for use in the query part of an ni name.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

[[Comments are included in double-square brackets, like this.]]

[[Note that the features here are less mature than the specification in the [nischeme] document. The intent is to develop these as required for the various use-cases as we go. If something from here appears to be as widely useful as the core ni scheme, then the authors are willing to move features from this document to the core document. We are also happy to incoroporate features to handle additional use-cases here if those arise.]]

2. Additional Algorithms

This section specifies two additional algorithms that MAY be used to handle truncated hashes and hashes calculated over dynamically changing objects.

For these optional algorithms, we establish a new IANA algorithm registry in Section 5.

2.1. Truncated Hashes

Message Digest algorithms are designed to provide protection against a collision attack. Due to the birthday paradox, this requires that the digest length be twice the length of a related encryption or authentication key to achieve the same work factor. Generally, hash function outputs will therefore be long, of the order of 256 bits (32 bytes raw, 43 bytes base64 encoded) or more. However, in some applications, strong collision resistance is not required, and ni names with shorter-hashes can be used without affecting security.

Different hash algorithm identifiers MUST be used for truncated hashes, that is, implementations MUST NOT accept digest values that are shorter than the (encoded) length for the specified hash

algorithm.

We define the sha-256-32 algorithm as being the leftmost 32 bits of output of the sha-256 algorithm and this algorithm is registered in Section 5.

[[Note: we probably need some discussion to pick a good truncated hash.]]

The following example includes an authority and uses a truncated variant of SHA-256.

ni://example.com/sha-256-32;B_K97zTtFuOhug27fke4_Q

[[Note: examples need to be checked.]]

2.2. Hashed Dynamic Content

The ni scheme involves calculating digest values over content objects. That works fine with static objects but is problematic for objects whose value is dynamically generated. In this section we define an algorithm that supports the same core "name-data integrity" service for dynamic objects. The basic idea is simply to include a hash of a public key in the ni name, and then for the dynamic object to be digitally signed with the corresponding private key. With a little work to handle the various useful formats, this allows a client that is presented with the ni name and the signed object to verify the binding between the name and the object data.

Note that the signature scheme used might or might not provide additional information, e.g. a name for the signer. Applications might benefit from that, but it is not required in order to provide the core "name-data integrity" service for dynamically generated objects.

Since there are a number of digital signing schemes that might be used, our approach is to define a new algorithm for the ni scheme that takes as input a specific public key encoded in a specific way, and runs that through a digest function. That is, the ni algorithm fields will specify both a public key algorithm and a digest algorithm, just as is done with digital signature algorithm identifiers.

Since it is possible that an ni algorithm might also be defined where the value contains an actual digital signature we need to be careful to ensure there is no ambiguity. However, since the lengths of signatures and hash outputs are (with current algorithms) always different, we could use that fact to disambigute between rsa-with-

sha256 meaning the value is a sha256 hash of an rsa public key and the alternative meaning the the value is an rsa-with-sha256 signature. However, we prefer to use a new registry (see Section 5 to ensure disambiguate these. The basic ni URI scheme requires algorithms to be chosen from the RFC 5698 registry, [RFC5698] for dynamic content, an algorithm from the registry defined here MUST be used.

We define one such algorithm, "pk-rsa-with-sha256" that takes an RSA public key as input, with the input bits formatted as a SubjectPublicKeyInfo as defined by RFC 5280. [RFC5280] Note that this does not mean that one cannot use e.g. PGP to sign the actual object. It means that if you do use PGP then in order to verify the name-data integrity, the client needs to extract the signer's PGP public key, then reformat that as a SubjectPublicKeyInfo and then run that through the sha-256 algorithm and make the relevant comparison.

3. Query String Paramaeters

This section defines query string parameters that MAY be used to indicate the type of content hashed or to specify additional locations from which the named content can be retrieved. We also define a way to specify how an encryption key MAY be included in an ni URI that allows for decryption of object content.

3.1. Digest with Content Type

The semantics of a digest being used to establish a secure reference from an authenticated source to an external source may be a function of associated meta data such as the content type. This data MAY be specified by means of a parameter:

ni:sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?ct=text/plain

The Content Type "ct" parameter specifies the MIME Content Type of the associated data as defined in [RFC4288]

3.2. Additional Locators

In addition to the algorithm for mapping an ni URI to an HTTP(S) URL specified in the ni scheme definition [nischeme], an ni name MAY provide information about additional locations from which the referenced content might be available. This is done via the inclusion of an "alt" or "alts" key in the query string that can supply more values for the authority field when constructing the HTTP or HTTPS URL. For example:

```
ni://example.com/
sha-
256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?alt=ni.example.net
```

The corresponding content might then also be retrieved from the URL:

```
http://ni.example.net/.well-known/ni/sha-256/
B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc
```

A ni name MAY specify multiple locations from which the content MAY be obtained:

```
ni:///
sha-
256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?alt=one.example.com&
alt=two.example.com
```

The above example asserts that the content might be retrieved from either of the following URIs:

```
http://one.example.com/.well-known/ni/sha-256/
B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc
```

```
http://two.example.com/.well-known/ni/sha-256/
B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc
```

The "alt" parameter means "use HTTP" and the "alts" parameter means use "HTTPS".

The alt and alts parameters are used to specify a possible means of resolving the referenced content. Multiple locator parameters MAY be used to specify alternative sources for accessing the content.

The alt and alts parameters take a single argument, the authority to be used for resolution. To permit the use of ni URIs in ASCII-only environments, the ASCII encoding (aka punycode) of the domain name MUST be used. [[Note sure if this is needed/correct.]]

3.3. Digest with Decryption Key

An ni name MAY provide a key for decrypting the referenced data. The use-case here is where the referenced data has be distributed (somehow) in ciphertext form, probably with little or no access control required (since the data is strongly encrypted) and where a client wishing to decrypt that data subsequently acquires an ni name for that data that provides the required decryption key.

Clearly, to be of any benefit, access to the ni name that includes

the decryption key MUST be controlled so that only the appropriate clients get access to the ni name and of course this ni name MUST be strongly protected via some (probably mutual) authentication and confidentiality service such as can be provided by TLS. [RFC5246]

ni///:sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?enc=aes-cbc:Fw3x20nEKfq6FDGzq7ttIQ

The "enc" specifier is used when the encrypted object consists of the ciphertext alone. The "menc" spcifier is used when the encrypted object consists of a MIME header containing metadata followed by the binary object encoding. [[Note: there may be more needed here.]]

The encryption specifiers both take an agrument of the form:

algorithm ":" base64url (key) [":" base64url (iv)]

Where

algorithm Is the algorithm used to encrypt the associated content

key Is the value of the cryptographic key

iv (optional) Is the value of the cryptographic Initialization Vector.

If the IV is not spcified for a block cipher mode that requires one, the IV MUST be prepended to the encrypted content.

[[Note: Actually the IV does not provide any additional security for this application but explaining the reason might be more effort than it is worth and what we really care about is saving bytes in the identifier, not the resulting data package.]

4. Security Considerations

[[We need to say when its safe, or not, to use truncated hashes.]]

[[More TBD no doubt.]]

5. IANA Considerations

5.1. Creation of ni additional algorithms registry

This specification creates a new IANA registry entitled "ni additional algorithms."

The policy for future assignments to the registry is "RFC Required".

The initial contents of the registry are:

Parameter	Meaning	Reference
sha-256-32	SHA-256 truncated to 32 bits	[RFC-THIS]
pk-rsa-with-sha256	Public key input to SHA-256	[RFC-THIS]

5.2. Creation of ni parameter registry

This specification creates a new IANA registry entitled "Named Information URI Parameter Definitions".

The policy for future assignments to the registry is "RFC Required".

The initial contents of the registry are:

Parameter	Meaning	Reference
ct	Content Type	[RFC-THIS]
alt	Additional HTTP Locator	[RFC-THIS]
alts	Additional HTTPS Locator	[RFC-THIS]
enc	Encryption Key	[RFC-THIS]
menc	Encryption Key	[RFC-THIS]

6. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5698] Kunz, T., Okunick, S., and U. Pordesch, "Data Structure for the Security Suitability of Cryptographic Algorithms (DSSC)", RFC 5698, November 2009.

[nischeme]
Farrell, S., Kutscher, D., Ohlman, B., and P. Hallam-Baker, "The Named Information (ni) URI Scheme: Core Syntax", draft-farrell-decade-ni-00 (work in progress), October 2011.

Authors' Addresses

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Rob Stradling
Comodo CA Ltd.

Email: rob.stradling@comodo.com

Stephen Farrell
Trinity College Dublin
Dublin, 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie

Dirk Kutscher
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: kutscher@neclab.eu

Boerje Ohlman
Ericsson
Stockholm S-16480
Sweden

Email: Borje.Ohlman@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: December 24, 2012

P. Hallam-Baker
Comodo Group Inc.
R. Stradling
Comodo CA Ltd.
S. Farrell
Trinity College Dublin
D. Kutscher
NEC
B. Ohlman
Ericsson
June 22, 2012

The Named Information (ni) URI Scheme: Optional Features
draft-hallambaker-decade-ni-params-03

Abstract

This document specifies optional things that one can do with "ni" URIs and related names. Those include an additional hash algorithm for handling dynamic content, some specific query-strong parameters for ni URIs and a mechanism for embedding ni URIs into QR codes.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 24, 2012.

Copyright Notice

Table of Contents

1. Introduction 3
2. Additional Algorithm 3
2.1. Hashed Dynamic Content 3
3. Query String Paramaeters 4
3.1. Additional Locators 4
3.2. Digest with Decryption Key 5
3.3. Wrapped URL 6
4. QR Codes 7
5. Security Considerations 7
6. IANA Considerations 7
6.1. Additional algorithm in ni Hash Algorithm Registry 7
6.2. Additional ni parameters 7
7. Normative References 7
Authors' Addresses 8

1. Introduction

The ni URI scheme [nischeme] supports extensibility in terms of the algorithm used to derive a value (normally, but not always a strong digest algorithm) and via support for a query-string thay can contain a list of key=value pairs. This document defines some uses for both of these extensibility points and creates IANA registries that can be used to register additional algorithms and key strings for use in the query part of an ni name. We [[will]] also define a way to embed ni names in QR codes.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

[[Comments are included in double-square brackets, like this.]]

[[Note that the features here are less mature than the specification in the [nischeme] document. The intent is to develop these as required for the various use-cases as we go. If something from here appears to be as widely useful as the core ni scheme, then the authors are willing to move features from this document to the core document. We are also happy to incoroporate features to handle additional use-cases here if those arise.]]

2. Additional Algorithm

This section specifies an additional algorithm that MAY be used to handle hashes calculated over dynamically changing objects.

2.1. Hashed Dynamic Content

The ni scheme involves calculating digest values over content objects. That works fine with static objects but is problematic for objects whose value is dynamically generated. In this section we define an algorithm that supports the same core "name-data integrity" service for dynamic objects. The basic idea is simply to include a hash of a public key in the ni name, and then for the dynamic object to be digitally signed with the corresponding private key. With a little work to handle the various useful formats, this allows a client that is presented with the ni name and the signed object to verify the binding between the name and the object data.

Note that the signature scheme used might or might not provide additional information, e.g. a name for the signer. Applications might benefit from that, but it is not required in order to provide the core "name-data integrity" service for dynamically generated

objects.

Since there are a number of digital signing schemes that might be used, our approach is to define a new algorithm for the ni scheme that takes as input a specific public key encoded in a specific way, and runs that through a digest function. That is, the ni algorithm fields will specify both a public key algorithm and a digest algorithm, just as is done with digital signature algorithm identifiers.

Since it is possible that an ni algorithm might also be defined where the value contains an actual digital signature we need to be careful to ensure there is no ambiguity. However, since the lengths of signatures and hash outputs are (with current algorithms) always different, we could use that fact to disambigute between rsa-with-sha256 meaning the value is a sha256 hash of an rsa public key and the alternative meaning the the value is an rsa-with-sha256 signature. However, we prefer to use a new algorithm (see Section 6 to disambiguate these.

We define one such algorithm, "pk-rsa-with-sha256" that takes an RSA public key as input, with the input bits formatted as a SubjectPublicKeyInfo as defined by [nischeme] Note that this does not mean that one cannot use e.g. PGP to sign the actual object. It means that if you do use PGP then in order to verify the name-data integrity, the client needs to extract the signer’s PGP public key, then reformat that as a SubjectPublicKeyInfo and then run that through the sha-256 algorithm and make the relevant comparison.

3. Query String Paramaeters

This section defines query string parameters that MAY be used to indicate the type of content hashed or to specify additional locations from which the named content can be retrieved. We also define a way to specify how an encryption key MAY be included in an ni URI that allows for decryption of object content.

3.1. Additional Locators

In addition to the algorithm for mapping an ni URI to an HTTP(S) URL specified in the ni scheme definition [nischeme], an ni name MAY provide information about additional locations from which the referenced content might be available. This is done via the inclusion of an "alt" or "alts" key in the query string that can supply more values for the authority field when constructing the HTTP or HTTPS URL. For example:

ni://example.com/
sha-
256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?alt=ni.example.net

The corresponding content might then also be retrieved from the URL:

http://ni.example.net/.well-known/ni/sha-256/
B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc

A ni name MAY specify multiple locations from which the content MAY be obtained:

ni:///
sha-
256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?alt=one.example.com&
alt=two.example.com

The above example asserts that the content might be retrieved from either of the following URIs:

http://one.example.com/.well-known/ni/sha-256/
B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc

http://two.example.com/.well-known/ni/sha-256/
B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc

The "alt" parameter means "use HTTP" and the "alts" parameter means use "HTTPS".

The alt and alts parameters are used to specify a possible means of resolving the referenced content. Multiple locator parameters MAY be used to specify alternative sources for accessing the content.

The alt and alts parameters take a single argument, the authority to be used for resolution. To permit the use of ni URIs in ASCII-only environments, the ASCII encoding (aka punycode) of the domain name MUST be used. [[Not sure if this is needed/correct.]]

3.2. Digest with Decryption Key

An ni name MAY provide a key for decrypting the referenced data. The use-case here is where the referenced data has be distributed (somehow) in ciphertext form, probably with little or no access control required (since the data is strongly encrypted) and where a client wishing to decrypt that data subsequently acquires an ni name for that data that provides the required decryption key.

Clearly, to be of any benefit, access to the ni name that includes

the decryption key MUST be controlled so that only the appropriate clients get access to the ni name and of course this ni name MUST be strongly protected via some (probably mutual) authentication and confidentiality service such as can be provided by TLS. [RFC5246]

ni///:sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?enc=aes-cbc:Fw3x20nEKfq6FDGzq7ttIQ

The "enc" specifier is used when the encrypted object consists of the ciphertext alone. The "menc" spcifier is used when the encrypted object consists of a MIME header containing metadata followed by the binary object encoding. [[Note: there may be more needed here.]]

The encryption specifiers both take an agrument of the form:

algorithm ":" base64url (key) [":" base64url (iv)]

Where

algorithm Is the algorithm used to encrypt the associated content

key Is the value of the cryptographic key

iv (optional) Is the value of the cryptographic Initialization Vector.

If the IV is not spcified for a block cipher mode that requires one, the IV MUST be prepended to the encrypted content.

[[Note: Actually the IV does not provide any additional security for this application but explaining the reason might be more effort than it is worth and what we really care about is saving bytes in the identifier, not the resulting data package.]

3.3. Wrapped URL

The "ni" URI form can usefully "wrap" an HTTP URL in order to provide a way for an HTTP client that has securely received an "ni" URI (e.g. embedded within some HTML received via a TLS session) to validate the referred-to content, at the same level of security as the embedding page. A good use for this might be to provide data integrity for javascript or other files referred to by an HTML page.

To achieve the above, we define the "url" parameter which allows for the inclusion of any URL within the query string. The intent is that the content accessed via that URL match the hash in the ni name.

[[TBD: say how to encode the URL]]

4. QR Codes

[[The idea of embedding ni names into QR codes has been floated. That seems like a fine thing to do, so we're likely to include text on that here in a future version.]]

5. Security Considerations

[[TBD for sure.]]

6. IANA Considerations

6.1. Additional algorithm in ni Hash Algorithm Registry

We request IANA to add a new entry to the hash algorithm registry created in [nischeme], Section 9.3, as follows:

ID: 7
Hash string name: pk-rsa-with-sha256
Value length: 256
Reference: [RFC-THIS]

6.2. Additional ni parameters

IANA is requested to add the following entries as defined above to the NI Hash Algorithm Name Registry.

Parameter	Meaning	Reference
alt	Additional HTTP Locator	[RFC-THIS]
alts	Additional HTTPS Locator	[RFC-THIS]
enc	Encryption Key	[RFC-THIS]
menc	Encryption Key	[RFC-THIS]
url	Wrapped URL	[RFC-THIS]

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[nischeme]
Farrell, S., Kutscher, D., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming things with hashes", draft-farrell-decade-ni-08 (work in progress), June 2012.

Authors' Addresses

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Rob Stradling
Comodo CA Ltd.

Email: rob.stradling@comodo.com

Stephen Farrell
Trinity College Dublin
Dublin, 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie

Dirk Kutscher
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: kutscher@neclab.eu

Boerje Ohlman
Ericsson
Stockholm S-16480
Sweden

Email: Borje.Ohlman@ericsson.com

DECADE
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

R. Alimi
Google
Y. Yang
Yale University
A. Rahman
InterDigital Communications, LLC
D. Kutscher
NEC
H. Liu
Yale University

October 31, 2011

DECADE Architecture
draft-ietf-decade-arch-04

Abstract

Content Distribution Applications (e.g., P2P applications) are widely used on the Internet and make up a large portion of the traffic in many networks. One technique to improve the network efficiency of these applications is to introduce storage capabilities within the networks; this is the capability to be provided by DECADE (DECoupled Application Data Enroute). This document presents an architecture for DECADE, discusses the underlying principles, and identifies core components and protocols for introducing in-network storage for these applications.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Table of Contents

1. Introduction 5
2. Functional Entities 6
2.1. DECADE Server 6
2.2. DECADE Client 6
2.3. DECADE Storage Provider 6
2.4. Content Provider Using DECADE 6
2.5. Content Consumer Using DECADE 7
2.6. Content Distribution Application 7
3. Protocol Flow 7
3.1. Overview 7
3.2. An Example 9
4. Architectural Principles 9
4.1. Decoupled Control/Metadata and Data Planes 10
4.2. Immutable Data Objects 11
4.3. Data Object Identifiers 12
4.4. Explicit Control 12
4.5. Resource and Data Access Control through User Delegation 12
5. System Components 13
5.1. Content Distribution Application 14
5.2. DECADE Server 16
5.3. Data Sequencing and Naming 18
5.4. Token-based Authentication and Resource Control 21
5.5. Discovery 22
6. DECADE Protocols 23
6.1. DECADE Resource Protocol (DRP) 23
6.2. Standard Data Transport (SDT) 26
7. Server-to-Server Protocols 29
7.1. Operational Overview 29
8. Potential Optimizations 30
8.1. Pipelining to Avoid Store-and-Forward Delays 30
8.2. Deduplication 31
9. Security Considerations 32
10. IANA Considerations 33
11. Acknowledgments 33
12. References 33
12.1. Normative References 33
12.2. Informative References 33
Appendix A. Appendix: Evaluation of Some Candidate Existing Protocols for DECADE DRP and SDT 34
A.1. HTTP 35
A.2. WEBDAV 37
A.3. CDMI 40
Appendix B. In-Network Storage Components Mapped to DECADE Architecture 42
B.1. Data Access Interface 43

B.2. Data Management Operations 43
B.3. Data Search Capability 43
B.4. Access Control Authorization 43
B.5. Resource Control Interface 43
B.6. Discovery Mechanism 43
B.7. Storage Mode 44
Authors' Addresses 44

1. Introduction

Content Distribution Applications are widely used on the Internet today to distribute data, and they contribute a large portion of the traffic in many networks. The DECADE architecture described in this document enables such applications to leverage in-network storage to achieve more efficient content distribution. Specifically, in many subscriber networks, it can be expensive to upgrade network equipment in the "last-mile", because it can involve replacing equipment and upgrading wiring at individual homes, businesses, and devices such as DSLAMs (Digital Subscriber Line Access Multiplexers) and CMTSs (Cable Modem Termination Systems) in remote locations. Therefore, it can be cheaper to upgrade the core infrastructure, which involves fewer components that are shared by many subscribers. See [I-D.ietf-decade-problem-statement] for a more complete discussion of the problem domain and general discussions of the capabilities to be provided by DECADE.

This document presents an architecture for providing in-network storage that can be integrated into Content Distribution Applications. The primary focus is P2P-based content distribution, but the architecture may be useful to other applications with similar characteristics and requirements. See [I-D.ietf-decade-reqs] for a definition of the target applications supported by DECADE.

The design philosophy of the DECADE architecture is to provide only the core functionalities that are needed for applications to make use of in-network storage. Focusing on only core functionalities, DECADE may be simpler and easier to support by service providers. If more complex functionalities are needed by a certain application or class of applications, it may be layered on top of DECADE.

DECADE will leverage existing data transport and application-layer protocols. The design is to work with a small set of alternative IETF protocols. In this document, we use "data transport" to refer to a protocol that is used to read data from and write data into DECADE in-network storage.

This document proceeds in two steps. First, it details the core architectural principles that we use to guide the DECADE design. Next, given these core principles, this document presents the core components of the DECADE architecture and identifies the usage of existing protocols and where there is a need for new protocol development.

This document does not define any new protocol. Instead, when identifying the need for a new protocol, it presents an abstract design including the necessary functions of that protocol (including

rationale) in order to guide future protocol development.

2. Functional Entities

This section defines the functional entities involved in a DECADE system. Functional entities can be classified as follows:

- A physical or logical component in the DECADE architecture: DECADE Client, DECADE Server, Content Distribution Application and Application End Point;
- Operator of a physical or logical component in the DECADE architecture: DECADE Storage Provider; and
- Source or sink of content distributed via the DECADE architecture: DECADE Content Provider, and DECADE Content Consumer.

2.1. DECADE Server

A DECADE server stores DECADE data inside the network, and thereafter manages both the stored data and access to that data. To reinforce that these servers are responsible for storage of raw data, this document also refers to them as storage servers.

2.2. DECADE Client

A DECADE client stores and retrieves data at DECADE Servers.

2.3. DECADE Storage Provider

A DECADE storage provider deploys and/or manages DECADE storage server(s) within a network. A storage provider may also own or manage the network in which the DECADE servers are deployed, but this is not mandatory.

A DECADE storage provider, possibly in cooperation with one or more network providers, determines deployment locations for DECADE servers and determines the available resources for each.

2.4. Content Provider Using DECADE

A content provider using DECADE accesses DECADE storage servers (by way of a DECADE client) to upload and manage data. Such a content provider can access one or more storage servers. Such a content provider may be a single process or a distributed application (e.g., in a P2P scenario), and may either be fixed or mobile.

2.5. Content Consumer Using DECADE

A content consumer using DECADE accesses DECADE storage servers (by way of a DECADE client). A content consumer can access one or more DECADE storage servers. A content consumer may be a single process or a distributed application (e.g., in a P2P scenario), and may either be fixed or mobile. An instance of a distributed application, such as a P2P application, may both provide content to and consume content from DECADE storage servers.

2.6. Content Distribution Application

A content distribution application (as a target application for DECADE as described in [I-D.ietf-decade-reqs]) is a distributed application designed for dissemination of a possibly-large data set to multiple consumers. Content Distribution Applications typically divide content into smaller blocks for dissemination.

The term Application Developer refers to the developer of a particular Content Distribution Application.

2.6.1. Application End-Point

An Application End-Point is an instance of a Content Distribution Application that makes use of DECADE server(s). A particular Application End-Point may be a DECADE Content Provider, a DECADE Content Consumer, or both. For example, an Application End-Point may be an instance of a video streaming client, or it may be the source providing the video to a set of clients.

An Application End-Point need not be actively transferring data with other Application End-Points to interact with the DECADE storage system. That is, an End-Point may interact with the DECADE storage servers as an offline activity.

3. Protocol Flow

3.1. Overview

The DECADE Architecture uses two protocols, as shown in Figure 1. First, the DECADE Resource Protocol is responsible for communication of access control and resource scheduling policies from DECADE Client to DECADE Server, as well as between DECADE Servers. The DECADE Architecture includes exactly one DRP for interoperability and a common format through which these policies can be communicated.

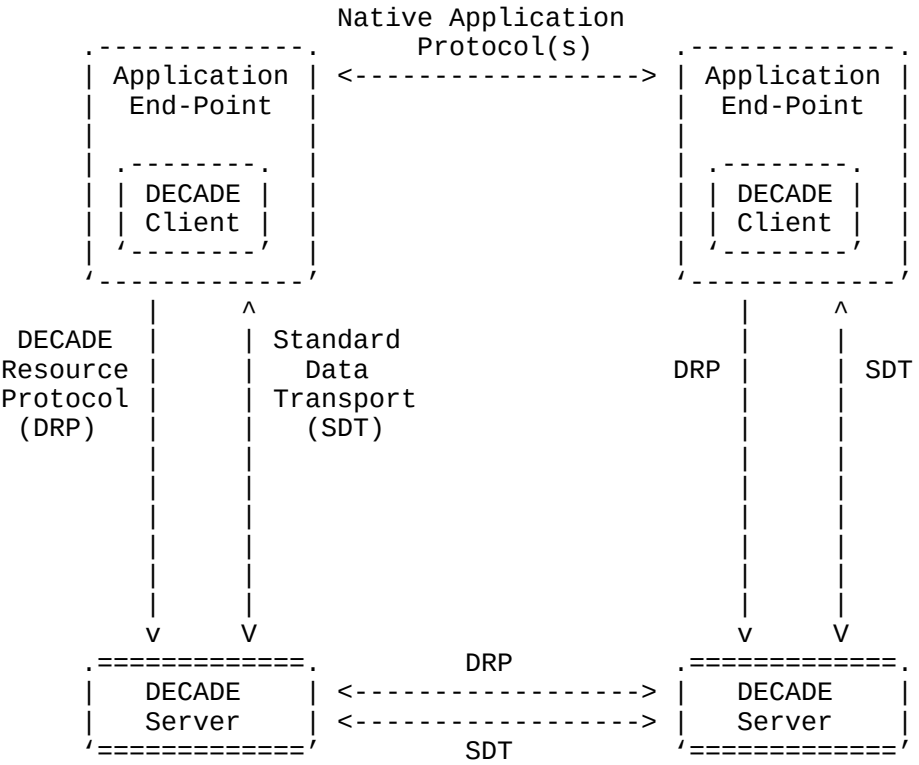


Figure 1: Generic Protocol Flow

Second, Standard Data Transport protocols (e.g., HTTP, WebDAV, or CDMI) are used to transfer data objects to and from a DECADE Server. The DECADE architecture may be used with multiple standard data transports.

Decoupling the protocols in this way allows DECADE to directly utilize existing standard data transports, as well as allowing both DECADE and DRP to evolve independently from data transports.

It is also important to note that the two protocols do not need to be separate on the wire. For example, DRP messages may be piggybacked within some extension fields provided by certain data transport protocols. In such a scenario, DRP is technically a data structure (transported by other protocols), but it can still be considered as a logical protocol that provides the services of configuring DECADE resource usage. Hence, this document considers SDT and DRP as two separate, logical functional components for clarity.

3.2. An Example

Before discussing details of the architecture, this section provides an example data transfer scenario to illustrate how the DECADE Architecture can be applied.

In this example, we assume that Application End-Point B (the receiver) is requesting a data object from Application End-Point A (the sender). Let S(A) denote A's DECADE storage server. There are multiple usage scenarios (by choice of the Content Distribution Application). For simplicity of introduction, we design the example to use only a single DECADE Server; Section 7 details a case when both A and B wish to employ DECADE Servers.

When an Application End-Point wishes to use its DECADE storage server, it provides a token (see Section 6.1.2 for details) to the other Application End-Point. The token is sent using the Content Distribution Application's native protocol.

The steps of the example are illustrated in Figure 2. First, B requests a data object from A using their native protocol. Next, A uses the DECADE Resource Protocol (DRP) to obtain a token from its DECADE storage server, S(A). A then provides the token to B (again, using their native protocol). Finally, B provides the token to S(A) via DRP, and requests and downloads the data object via a Standard Data Transport (SDT).

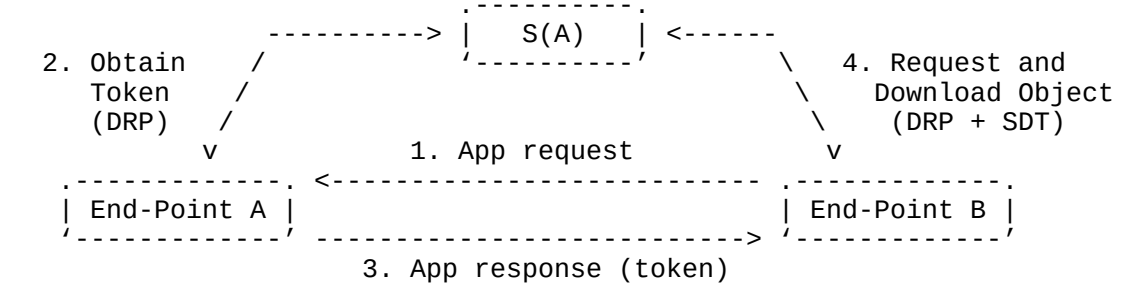


Figure 2: Download from Storage Server

4. Architectural Principles

We identify the following key principles.

4.1. Decoupled Control/Metadata and Data Planes

The DECADE infrastructure is intended to support multiple content distribution applications. A complete content distribution application implements a set of control and management functions including content search, indexing and collection, access control, ad insertion, replication, request routing, and QoS scheduling. An observation of DECADE is that different content distribution applications can have unique considerations designing the control and signaling functions:

- Metadata Management Scheme: Traditional file systems provide a standard metadata abstraction: a recursive structure of directories to offer namespace management; each file is an opaque byte stream. In content distribution, applications may use different metadata management schemes. For example, one application may use a sequence of blocks (e.g., for file sharing), while another application may use a sequence of frames (with different sizes) indexed by time.

- Resource Scheduling Algorithms: a major competitive advantage of many successful P2P systems is their substantial expertise in achieving highly efficient utilization of peer and infrastructural resources. For instance, many live P2P systems have their specific algorithms in constructing topologies to achieve low-latency, high-bandwidth streaming. They continue to fine-tune such algorithms.

Given the diversity of control-plane functions, in-network storage should export basic mechanisms and allow as much flexibility as possible to the control planes to implement specific policies. This conforms to the end-to-end systems principle and allows innovation and satisfaction of specific business goals.

Decoupling control plane and data plane is not new. For example, OpenFlow is an implementation of this principle for Internet routing, where the computation of the forwarding table and the application of the forwarding table are separated. Google File System [GoogleFileSystem] applies the principle to file system design, by utilizing the Master to handle the meta-data management, and the Chunk Servers to handle the data plane functions (i.e., read and write of chunks of data). NFSv4.1's pNFS extension [RFC5661] also implements this principle.

Note that applications may have different Data Plane implementations in order to support particular requirements (e.g., low latency). In order to provide interoperability, the DECADE architecture does not intend to enable arbitrary data transport protocols. However, the

architecture may allow for more-than-one data transport protocols to be used.

Also note that although an application's existing control plane functions remain implemented within the application, the particular implementation may need to be adjusted to support DECADE.

4.2. Immutable Data Objects

A property of bulk contents to be broadly distributed is that they typically are immutable -- once a piece of content is generated, it is typically not modified. It is not common that bulk contents such as video frames and images need to be modified after distribution.

Many content distribution applications divide content objects into blocks for two reasons: (1) multipath: different blocks may be fetched from different content sources in parallel, and (2) faster recovery and verification: individual blocks may be recovered and verified. Typically, applications use a block size larger than a single packet in order to reduce control overhead.

Common applications using the aforementioned data model include P2P streaming (live and video-on-demand) and P2P file-sharing. However, other additional types of applications may match this model.

DECADE adopts a design in which immutable data objects may be stored at a storage server. Applications may consider existing blocks as DECADE data objects, or they may adjust block sizes before storing in a DECADE server.

Focusing on immutable data blocks in the data plane can substantially simplify the data plane design, since consistency requirements can be relaxed. It also allows effective reuse of data blocks and de-duplication of redundant data.

Depending on its specific requirements, an application may store data in DECADE servers such that each data object is completely self-contained (e.g., a complete, independently decodable video segment). An application may also divide data into chunks that require application level assembly. The DECADE architecture and protocols are agnostic to the nature of the data objects and do not specify a fixed size for them.

Note that immutable content may still be deleted. Also note that immutable data blocks do not imply that contents cannot be modified. For example, a meta-data management function of the control plane may associate a name with a sequence of immutable blocks. If one of the blocks is modified, the meta-data management function changes the

mapping of the name to a new sequence of immutable blocks.

Throughout this document, all the data objects/blocks are referred as immutable data objects/blocks.

4.3. Data Object Identifiers

Objects that are stored in a DECADE storage server can be accessed by DECADE content consumers by a resource identifier that has been assigned within a certain application context.

Because a DECADE content consumer can access more than one storage server within a single application context, a data object that is replicated across different storage servers managed by a DECADE storage provider, can be accessed by a single identifier.

Note that since data objects are immutable, it is possible to support persistent identifiers for data objects.

4.4. Explicit Control

To support the functions of an application’s control plane, applications must be able to know and control which data is stored at particular locations. Thus, in contrast with content caches, applications are given explicit control over the placement (selection of a DECADE server), deletion (or expiration policy), and access control for stored data.

Consider deletion/expiration policy as a simple example. An application may require that a DECADE server store content for a relatively short period of time (e.g., for live-streaming data). Another application may need to store content for a longer duration (e.g., for video-on-demand).

4.5. Resource and Data Access Control through User Delegation

DECADE provides a shared infrastructure to be used by multiple tenants of multiple content distribution applications. Thus, it needs to provide both resource and data access control.

4.5.1. Resource Allocation

There are two primary interacting entities in the DECADE architecture. First, Storage Providers control where DECADE storage servers are provisioned and their total available resources. Second, Applications control data transfers amongst available DECADE servers and between DECADE servers and end-points. A form of isolation is required to enable concurrently-running Applications to each

explicitly manage its own content and share of resources at the available servers.

The Storage Provider delegates the management of the resources at a DECADE server to one or more applications. Applications are able to explicitly and independently manage their own shares of resources.

4.5.2. User Delegations

Storage providers have the ability to explicitly manage the entities allowed to utilize the resources at a DECADE server. This capability is needed for reasons such as capacity-planning and legal considerations in certain deployment scenarios.

To provide a scalable way to manage applications granted resources at a DECADE server, we consider an architecture that adds a layer of indirection. Instead of granting resources to an application, the DECADE server grants a share of the resources to a user. The user may in turn share the granted resources amongst multiple applications. The share of resources granted by a storage provider is called a User Delegation.

As a simple example, a DECADE Server operated by an ISP may be configured to grant each ISP Subscriber 1.5 Mbps of bandwidth. The ISP Subscriber may in turn divide this share of resources amongst a video streaming application and file-sharing application which are running concurrently.

In general, a User Delegation may be granted to an end-user (e.g., an ISP subscriber), a Content Provider, or an Application Provider. A particular instance of an application may make use of the storage resources:

- granted to the end-user (with the end-user’s permission),
- granted to the Content Provider (with the Content Provider’s permission), and/or
- granted to the Application Provider.

5. System Components

The primary focus of this document is the architectural principals and the system components that implement them. While certain system components might differ amongst implementations, the document details the major components and their overall roles in the architecture.

To keep the scope narrow, we only discuss the primary components related to protocol development. Particular deployments may require additional components (e.g., monitoring and accounting at a DECADE server), but they are intentionally omitted from this document.

5.1. Content Distribution Application

Content Distribution Applications have many functional components. For example, many P2P applications have components and algorithms to manage overlay topology management, piece selection, etc. In supporting DECADE, it may be advantageous for an application developer to consider DECADE in the implementation of these components. However, in this architecture document, we focus on the components directly employed to support DECADE.

Figure 3 illustrates the components discussed in this section from the perspective of a single Application End-Point and their relation to DECADE.

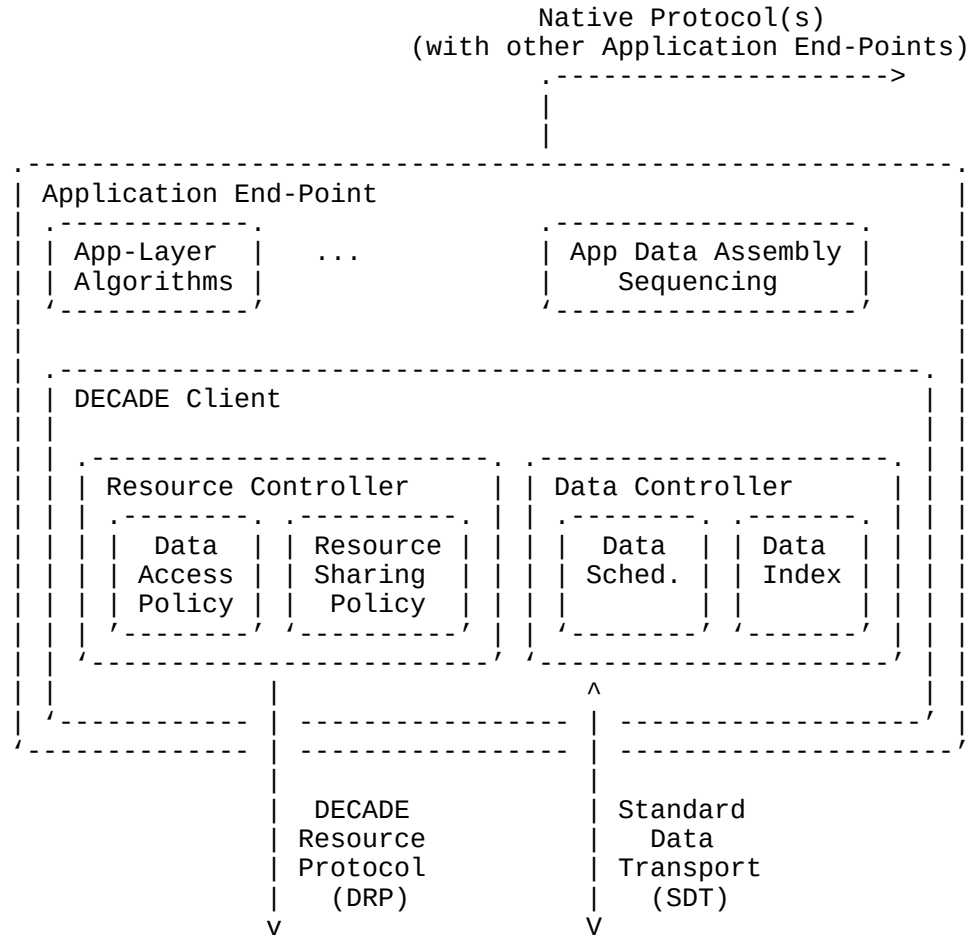


Figure 3: Application Components

5.1.1. Data Assembly

DECADE is primarily designed to support applications that can divide distributed contents into data objects. To accomplish this, applications include a component responsible for creating the individual data objects before distribution and then re-assembling data objects at the Content Consumer. We call this component Application Data Assembly. The specific implementation is entirely decided by the application.

In producing and assembling the data objects, two important considerations are sequencing and naming. The DECADE architecture assumes that applications implement this functionality themselves.

See Section 5.3 for further discussion.

5.1.2. Native Protocols

Applications may still use existing protocols. In particular, an application may reuse existing protocols primarily for control/signaling. However, an application may still retain its existing data transport protocols, in addition to DECADE as the data transport protocol. This can be important for applications that are designed to be highly robust (e.g., if DECADE servers are unavailable).

5.1.3. DECADE Client

An application may be modified to support DECADE. We call the layer providing the DECADE support to an application the DECADE Client. It is important to note that a DECADE Client need not be embedded into an application. It could be implemented alone, or could be integrated in other entities such as network devices themselves.

5.1.3.1. Resource Controller

Applications may have different Resource Sharing Policies and Data Access Policies to control their resource and data in DECADE servers. These policies can be existing policies of applications (e.g., tit-for-tat) or custom policies adapted for DECADE. The specific implementation is decided by the application.

5.1.3.2. Data Controller

DECADE is designed to decouple the control and the data transport of applications. Data transport between applications and DECADE servers uses standard data transport protocols. A Data Scheduling component schedules data transfers according to network conditions, available DECADE Servers, and/or available DECADE Server resources. The Data Index indicates data available at remote DECADE servers. The Data Index (or a subset of it) may be advertised to other Application End-Points. A common use case for this is to provide the ability to locate data amongst a distributed set of Application End-Points (i.e., a data search mechanism).

5.2. DECADE Server

A DECADE Server stores data from Application End-Points, and provides control and access of those data to Application End-Points. Note that a DECADE Server is not necessarily a single physical machine, it could also be implemented as a cluster of machines.

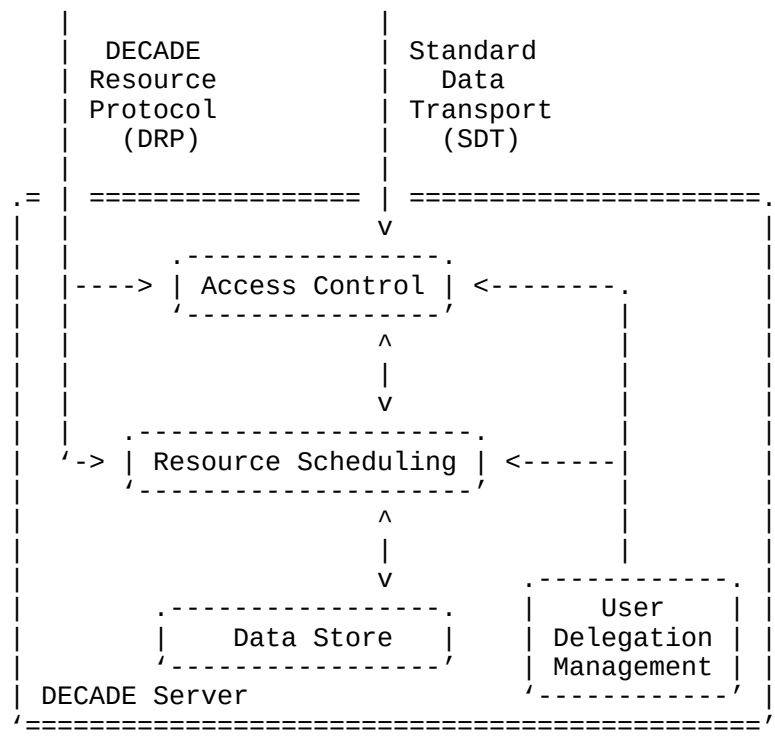


Figure 4: DECADE Server Components

5.2.1. Access Control

An Application End-Point can access its own data or other Application End-Point's data (provided sufficient authorization) in DECADE servers. Application End-Points may also authorize other End-Points to store data. If an access is authorized by an Application End-Point, the DECADE Server will provide access.

Note that even if a request is authorized, it may still fail to complete due to insufficient resources by either the requesting Application End-Point, the providing Application End-Point, or the DECADE Server itself.

5.2.2. Resource Scheduling

Applications may apply their existing resource sharing policies or use a custom policy for DECADE. DECADE servers perform resource scheduling according to the resource sharing policies indicated by Application End-Points as well as configured User Delegations.

5.2.3. Data Store

Data from applications may be stored at a DECADE Server. Data can be deleted from storage either explicitly or automatically (e.g., after a TTL expiration). It may be possible to perform optimizations in certain cases, such as avoiding writing temporary data (e.g., live streaming) to persistent storage, if appropriate storage hints are supported by the SDT.

5.3. Data Sequencing and Naming

In order to provide a simple and generic interface, the DECADE Server is only responsible for storing and retrieving individual data objects. Furthermore, DECADE uses its own simple naming scheme that provides uniqueness (with high probability) between data objects, even across multiple applications.

5.3.1. DECADE Data Object Naming Scheme

The name of a data object is derived from the hash over the data object's content (the raw bytes), which is made possible by the fact that DECADE objects are immutable. This scheme multiple appealing properties:

- Simple integrity verification
- Unique names (with high probability)
- Application independent, without a new IANA-maintained registry

The DECADE naming scheme also includes a "type" field, the "type" identifier indicates that the name is the hash of the data object's content and the particular hashing algorithm used. This allows DECADE to evolve by either changing the hashing algorithm (e.g., if security vulnerabilities with an existing hashing algorithm are discovered), or moving to a different naming scheme altogether.

The specific format of the name (e.g., encoding, hash algorithms, etc) is out of scope of this document, and left for protocol specification.

Another advantage of this scheme is that a DECADE client knows the name of a data object before it is completely stored at the DECADE server. This allows for particular optimizations, such as advertising data object while the data object is being stored, removing store-and-forward delays. For example, a DECADE client A may simultaneously begin storing an object to a DECADE server, and advertise that the object is available to DECADE client B. If it is

supported by the DECADE server, client B may begin downloading the object before A is finished storing the object.

In certain scenarios (e.g., where a DECADE client has limited computational power), it may be advantageous to offload the computation of a data object's name to the DECADE Server. This possibility is not considered in the current architecture, but may be a topic for future extensions.

5.3.2. Application Usage

Recall from Section 5.1.1 that an Application typically includes its own naming and sequencing scheme. It is important to note that the DECADE naming format does not attempt to replace any naming or sequencing of data objects already performed by an Application; instead, the DECADE naming is intended to apply only to data objects referenced at the DECADE layer.

DECADE names are not necessarily correlated with the naming or sequencing used by the Application using a DECADE client. The DECADE client is expected to maintain a mapping from its own data objects and their names to the DECADE data objects and names. Furthermore, the DECADE naming scheme implies no sequencing or grouping of objects, even if this is done at the application layer.

Not only does an Application retain its own naming scheme, it may also decide the sizes of data objects to be distributed via DECADE. This is desirable since sizes of data objects may impact Application performance (e.g., overhead vs. data distribution delay), and the particular tradeoff is application-dependent.

5.3.3. Application Usage Example

To illustrate these properties, this section presents multiple examples.

5.3.3.1. Application with Fixed-Size Chunks

Similar to the example in Section 5.1.1, consider an Application in which each individual application-layer segment of data is called a "chunk" and has a name of the form: "CONTENT_ID:SEQUENCE_NUMBER". Furthermore, assume that the application's native protocol uses chunks of size 16KB.

Now, assume that this application wishes to make use of DECADE, and assume that it wishes to store data to DECADE servers in data objects of size 64KB. To accomplish this, it can map a sequence of 4 chunks into a single DECADE object, as shown in Figure 5.

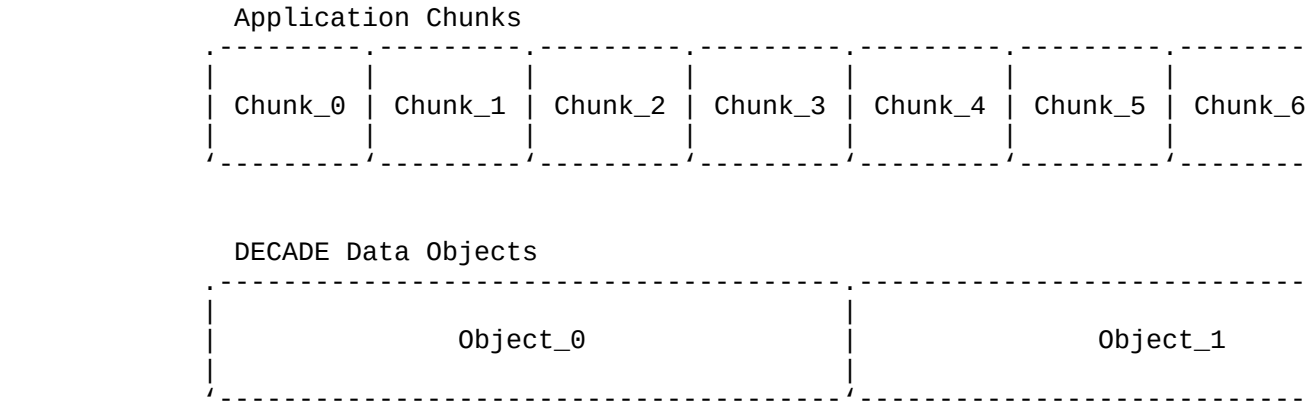


Figure 5: Mapping Application Chunks to DECADE Data Objects

In this example, the Application might maintain a logical mapping that is able to determine the name of a DECADE data object given the chunks contained within that data object. The name might be learned from either the original source, another endpoint with which the it is communicating, a tracker, etc.

It is important to note that as long as the data contained within each sequence of chunks is unique, the corresponding DECADE data objects have unique names. This is desired, and happens automatically if particular Application segments the same stream of data in a different way, including different chunk size sizes or different padding schemes.

5.3.3.2. Application with Continuous Streaming Data

Next, consider an Application whose native protocol retrieves a continuous data stream (e.g., an MPEG2 stream) instead of downloading and redistributing chunks of data. Such an application could segment the continuous data stream to produce either fixed-sized or variable-sized DECADE data objects.

Figure 6 shows how a video streaming application might produce variable-sized DECADE data objects such that each DECADE data object contains 10 seconds of video data.

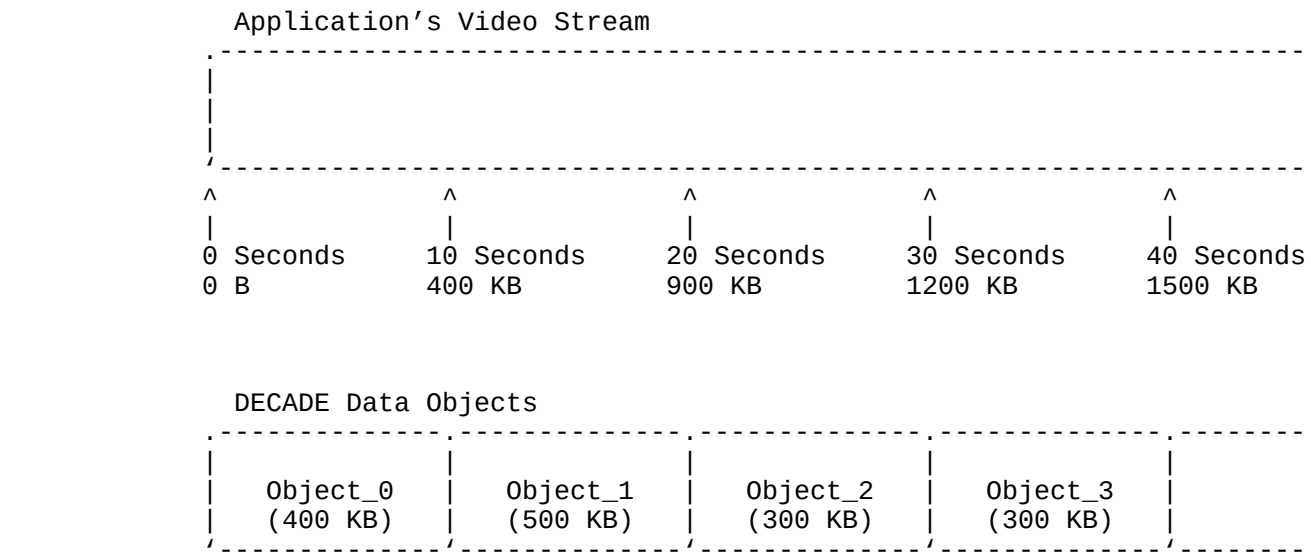


Figure 6: Mapping a Continuous Data Stream to DECADE Data Objects

Similar to the previous example, the Application might maintain a mapping that is able to determine the name of a DECADE data object given the time offset of the video chunk.

5.4. Token-based Authentication and Resource Control

A primary use case for DECADE is a DECADE Client authorizing other DECADE Clients to store or retrieve data objects from its DECADE storage. To support this, DECADE uses a token-based authentication scheme.

In particular, an entity trusted by a DECADE Client generates a digitally-signed token with particular properties (see Section 6.1.2 for details). The DECADE Client distributes this token to other DECADE Clients which then use the token when sending requests to the DECADE Server. Upon receiving a token, the DECADE Server validates the signature and the operation being performed.

This is a simple scheme, but has multiple important advantages over an alternate approach in which a DECADE Client explicitly manipulates an Access Control List (ACL) associated with each DECADE data object. In particular, it has the following advantages when applied to DECADE's target applications:

- Authorization policies are implemented within the Application; an Application explicitly controls when tokens are generated and to

whom they are distributed.

- Fine-grained access and resource control can be applied to data objects; see Section 6.1.2 for the list of restrictions that can be enforced with a token.

- There is no messaging between a DECADE Client and DECADE Server to manipulate data object permissions. This can simplify, in particular, Applications which share data objects with many dynamic peers and need to frequently adjust access control policies attached to DECADE data objects.

- Tokens can provide anonymous access, in which a DECADE Server does not need to know the identity of each DECADE Client that accesses it. This enables a DECADE Client to send tokens to DECADE Clients in other administrative or security domains, and allow them to read or write data objects from its DECADE storage.

It is important to note that, in addition to DECADE Clients applying access control policies to DECADE data objects, the DECADE Server may be configured to apply additional policies based on user, object, geographic location, etc. Defining such policies is out of scope for DECADE, but in such a case, a DECADE Client may be denied access even though it possess a valid token.

5.5. Discovery

DECADE includes a discovery mechanism through which DECADE clients locate an appropriate DECADE Server. [I-D.ietf-decade-reqs] details specific requirements of the discovery mechanism; this section discusses how they relate to other principles outlined in this document.

A discovery mechanism allows a DECADE client to determine an IP address or some other identifier that can be resolved to locate the server for which the client will be authorized to generate tokens (via DRP). (Note that the discovery mechanism may also result in an error if no such DECADE servers can be located.) After discovering one or more DECADE servers, a DECADE client may distribute load and requests across them (subject to resource limitations and policies of the DECADE servers themselves) according to the policies of the Application End-Point in which it is embedded.

The particular protocol used for discovery is out of scope of this document, but any specification will re-use standard protocols wherever possible.

It is important to note that the discovery mechanism outlined here

does not provide the ability to locate arbitrary DECADE servers to which a DECADE client might obtain tokens from others. To do so requires application-level knowledge, and it is assumed that this functionality is implemented in the Content Distribution Application, or if desired and needed, as an extension to this DECADE architecture.

6. DECADE Protocols

This section presents the DECADE Resource Protocol (DRP) and the Standard Data Transport (SDT) in terms of abstract protocol interactions that are intended to mapped to specific protocols. Note that while the protocols are logically separate, DRP is specified as being carried through extension fields within an SDT (e.g., HTTP headers).

The DRP is the protocol used by a DECADE client to configure the resources and authorization used to satisfy requests (reading, writing, and management operations concerning DECADE objects) at a DECADE server. The SDT is used to send the operations to the DECADE server. Necessary DRP metadata is supplied using mechanisms in the SDT that are provided for extensibility (e.g., additional request parameters or extension headers).

6.1. DECADE Resource Protocol (DRP)

DRP provides configuration of access control and resource sharing policies on DECADE servers. A content distribution application, e.g., a live P2P streaming session, MAY employ several DECADE servers, for instance, servers in different operator domains, and DRP allows one instance of such an application, e.g., an application endpoint, to apply access control and resource sharing policies on each of them.

6.1.1. Controlled Resources

On a single DECADE server, the following resources may be managed:

communication resources: DECADE servers have limited communication resources in terms of bandwidth (upload/download) but also in terms of number of connected clients (connections) at a time.

storage resources: DECADE servers have limited storage resources.

6.1.2. Access and Resource Control Token

A token includes the following fields:

- Permitted operations (e.g., read, write)
- Permitted objects (e.g., names of data objects that may be read or written)
- Permitted clients (e.g., as indicated by IP address or other identifier) that may use the token
- Expiration time
- Priority for bandwidth given to requested operation (e.g., a weight used in a weighted bandwidth sharing scheme)
- Amount of data that may be read or written

The particular format for the token is out of scope of this document.

The tokens are generated by a trusted entity at the request of a DECADE Client. It is out of scope of this document to identify which entity serves this purpose, but examples include the DECADE Client itself, a DECADE Server trusted by the DECADE Client, or another server managed by a Storage Provider trusted by the DECADE Client.

Upon generating a token, a DECADE Client may distribute it to another DECADE Client (e.g., via their native Application protocol). The receiving DECADE Client may then connect to the sending DECADE Client's DECADE Server and perform any operation permitted by the token. The token must be sent along with the operation. The DECADE Server validates the token to identify the DECADE Client that issued it and whether the requested operation is permitted by the contents of the token. If the token is successfully validated, the DECADE Server applies the resource control policies indicated in the token while performing the operation.

It is possible for DRP to allow tokens to apply to a batch of operations to reduce communication overhead required between DECADE Clients.

DRP may also define tokens to include a unique identifier to allow a DECADE Server to detect when a token is used multiple times.

6.1.3. Status Information

DRP provides a request service for status information that DECADE clients can use to request information from a DECADE server.

status information per application context on a specific server:
Access to such status information requires client authorization, i.e., DECADE clients need to be authorized to access status information for a specific application context. This authorization (and the mapping to application contexts) is based on the user delegation concept as described in Section 4.5. The following status information elements can be obtained:

* list of associated objects (with properties)

* resources used/available

* list of servers to which objects have been distributed (in a certain time-frame)

* list of clients to which objects have been distributed (in a certain time-frame)

For the list of servers/clients to which objects have been distributed to, the DECADE server can decide on time bounds for which this information is stored and specify the corresponding time frame in the response to such requests. Some of this information can be used for accounting purposes, e.g., the list of clients to which objects have been distributed.

access information per application context on a specific server:
Access information can be provided for accounting purposes, for example, when application service providers are interested to maintain access statistics for resources and/or to perform accounting per user. Again, access to such information requires client authorization based on the user delegation concept as described in Section 4.5. The following access information elements can be requested:

* what objects have been accessed how many times

* access tokens that a server as seen for a given object

The DECADE server can decide on time bounds for which this information is stored and specify the corresponding time frame in the response to such requests.

6.1.4. Object Attributes

Objects that are stored on a DECADE server may have associated attributes (in addition to the object identifier and the actual content) that relate to the data storage and its management. These attributes may be used by the DECADE server (and possibly the underlying storage system) to perform specialized processing or handling for the data object, or to attach related DECADE server or storage-layer properties to the data object. These attributes have a scope local to a DECADE server. In particular, these attributes are not applied to a DECADE server or client to which a data object is copied.

Depending on authorization, DECADE clients may get or set such attributes. This authorization (and the mapping to application contexts) is based on the user delegation concept as described in Section 4.5. The DECADE architecture does not limit the set of permissible attributes, but rather specifies a set of baseline attributes that SHOULD be supported by implementations.

Suggested attributes are the following:

TTL: TTL of the object as an absolute time value

object size: in bytes

MIME type

access statistics: how often the object has been accessed (and what tokens have been used)

It is important to note that the Object Attributes defined here are distinct from application metadata (see Section 4.1). Application metadata is custom information that an application may wish to associate with a data object to understand its semantic meaning (e.g., whether it is video and/or audio, its playback length in time, or its index in a stream). If an application wishes to store such metadata persistently within DECADE, it can be stored within data objects themselves.

6.2. Standard Data Transport (SDT)

A DECADE server provide a data access interface, and SDT is used to write objects to a server and to read (download) objects from a server. Semantically, SDT is a client-server protocol, i.e., the DECADE server always responds to client requests.

An SDT used in DECADE SHOULD offer a transport mode that provides

confidentiality and integrity.

6.2.1. Writing/Uploading Objects

To write an object, a client first generates the object's name (see Section 5.3), and then uploads the object to a DECADE server and supplies the generated name. The name can be used to access (download) the object later, e.g., the client can pass the name as a reference to other client that can then refer to the object.

DECADE objects can be self-contained objects such as multimedia resources, files etc., but also chunks, such as chunks of a P2P distribution protocol that can be part of a containing object or a stream.

A server MAY accept download requests for an object that is still being uploaded.

The application that originates the objects MUST generate DECADE object names according to the naming specification in Section 5.3. The naming scheme provides that the name is unique. DECADE clients (as parts of application entities) upload a named object to a server, and a DECADE server MUST NOT change the name. It MUST be possible for downloading clients, to access the object using the original name. A DECADE server MAY verify the integrity and other security properties of uploaded objects.

In the following we provide an abstract specification of the upload operation that we name 'PUT METHOD'. See Appendix A.1 for an example how this could be mapped to HTTP.

Method PUT:

Parameters:

NAME: The naming of the object according to Section 5.3

OBJECT: The object itself. The protocol MUST provide transparent binary object transport.

Description: The PUT method is used by a DECADE client to upload an object with an associated name 'NAME' to a DECADE server.

RESPONSES: The DECADE server MUST respond with one the following response messages:

CREATED: The object has been uploaded successfully and is now available under the specified name.

ERRORs:

VALIDATION_FAILED: The contents of the data object received by the DECADE server did not match the provided name (i.e., hash validation failed).

PERMISSION_DENIED: The DECADE client lacked sufficient permission to store the object.

Specifics regarding error handling, including additional error conditions, precedence for returned errors and its relation with server policy, are deferred to eventual protocol specification.

6.2.2. Downloading Objects

A DECADE client can request named objects from a DECADE server. In the following, we provide an abstract specification of the download operation that we name 'GET METHOD'. See Appendix A.1 for an example how this could be mapped to HTTP.

Method GET:

Parameters:

NAME: The naming of the object according to Section 5.3.

Description: The GET method is used by a DECADE client to download an object with an associated name 'NAME' from a DECADE server.

RESPONSES: The DECADE server MUST respond with one the following response messages:

OK: The request has succeeded, and an entity corresponding to the requested resource is sent in the response.

ERRORs:

NOT_FOUND: The DECADE server has not found anything matching the request object name.

PERMISSION_DENIED: The DECADE client lacked sufficient permission to read the object.

NOT_AVAILABLE: The data object exists but is currently unavailable for download (e.g., due to server policy).

Specifics regarding error handling, including additional error conditions (e.g. overload), precedence for returned errors and its relation with server policy, are defered to eventual protocol specification.

7. Server-to-Server Protocols

An important feature of DECADE is the capability for one DECADE server to directly download objects from another DECADE server. This capability allows Applications to directly replicate data objects between servers without requiring end-hosts to use uplink capacity to upload data objects to a different DECADE server.

To support this functionality, DECADE uses the DRP and SDT to support operations directly between servers. DECADE servers are not assumed to trust each other nor are configured to do so. All data operations are performed on behalf of DECADE clients via explicit instruction. Note, however, that the objects being processed do not necessarily have to originate or terminate at the DECADE client (i.e. the object may be limited to being exchanged between DECADE servers even if the instruction is triggered by the client). DECADE clients thus must be able to indicate to a DECADE server the following additional parameters:

- which remote DECADE server(s) to access;
- the operation to be performed (e.g. PUT, GET); and
- Credentials indicating permission to perform the operation at the remote DECADE server.

In this way, a DECADE server acts as a proxy for a DECADE client, and a DECADE client may instantiate requests via that proxy. The operations are performed as if the original requester had its own DECADE client co-located with the DECADE server. It is this mode of operation that provides substantial savings in uplink capacity. Note that this mode of operation may also be triggered by an administrative/management application outside the DECADE architecture.

7.1. Operational Overview

DECADE’s server-to-server support is focused on reading and writing data objects between DECADE servers. A DECADE GET or PUT request MAY

supply the following additional parameters:

REMOTE_SERVER: Address of the remote DECADE server. The format of the address is out-of-scope of this document.

REMOTE_USER: The account at the remote server from which to retrieve the object (for a GET), or in which the object is to be stored (for a PUT).

TOKEN: Credentials to be used at the remote server.

These parameters are used by the DECADE server to instantiate a request to the specified remote server. It is assumed that the data object referred to at the remote server is the same as the original request. It is also assumed that the operation performed at the remote server is the same as the operation in the original request. Note that object attributes (see Section 6.1.4) may also be specified in the request to the remote server.

Note that when a DECADE client invokes a request a DECADE server with these additional parameters, it is giving the DECADE server permission to act (proxy) on its behalf. Thus, it would be wise for the supplied token to have narrow privileges (e.g., limited to only the necessary data objects) or validity time (e.g., a small expiration time).

In the case of a GET operation, the DECADE server is to retrieve the data object from the remote server using the specified credentials (via a GET request to the remote server), and then optionally return the object to a client. In the case of a PUT operation, the DECADE server is to store the object to the remote server using the specified credentials (via a PUT request to the remote server). The object may optionally be uploaded from the client or may already exist at the proxying server.

8. Potential Optimizations

As suggestions for the protocol design and eventual implementations, we discuss particular optimizations that are enabled by the DECADE Architecture discussed in this document.

8.1. Pipelining to Avoid Store-and-Forward Delays

A DECADE server may choose to not fully store an object before beginning to serve it. For example, when serving a GET request, instead of waiting for the complete data to arrive from a remote server or DECADE client, a DECADE server may forward received data

bytes as they come in. This pipelining mode reduces store-and-forward delays, which could be substantial for large objects. A similar behavior could be used for PUT.

8.2. Deduplication

A common concern amongst Storage Providers is the total volume of data that needs to be stored. An optimization frequently applied in existing storage systems is de-duplication, which attempts to avoid storing identical data multiple times. A DECADE Server implementation may internally perform de-duplication of data on disk. The DECADE architecture enables additional forms of de-duplication.

Note that these techniques may impact protocol design. Discussions of whether or not they should be adopted is out of the scope of this document.

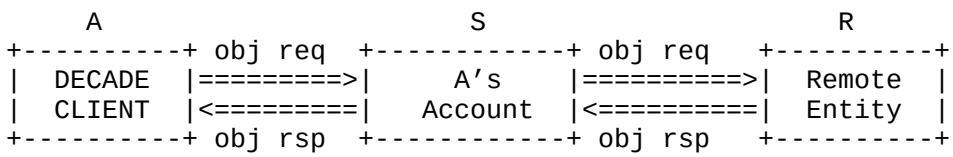
8.2.1. Traffic De-duplication

8.2.1.1. Rationale

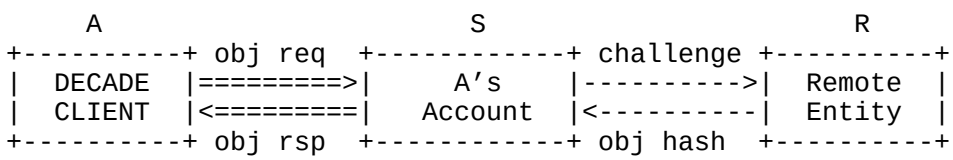
When a DECADE client (A) indicates its DECADE account on a DECADE server (S) to fetch an object from a remote entity (R) (a DECADE server or DECADE client) and if the object is already stored locally in S, S may perform Traffic De-duplication. This means that S does not download the object from R, in order to save network traffic. In particular, S performs a challenge to make sure that the remote entity R actually has the object and then replies with its local object copy directly.

8.2.1.2. An Example

As shown in Figure 7, without Traffic De-duplication, unnecessary transfer of an object from R to S may happen, if the server S already has the object requested by A. If Traffic De-duplication is enabled, S only needs to challenge R to verify that it does have the data to avoid data-stealing attacks.



(a) Without Traffic De-duplication



(b) With Traffic De-duplication

Figure 7

8.2.1.3. HTTP Compatibility of Challenge

How to integrate traffic de-duplication with HTTP is shown in Appendix A.1.3.

8.2.2. Cross-Server Storage De-duplication

The same object might be uploaded multiple times to different DECADE servers. For storage efficiency, storage providers may desire that a single object be stored on one or a few servers. They might implement an internal mechanism to achieve the goal, for example, by redirecting requests to proper servers. DECADE supports the redirection of DECADE client requests to support further cross-server storage de-duplication.

9. Security Considerations

In general, the security considerations mentioned in [I-D.ietf-decade-problem-statement] apply to this document as well.

In addition, it should be noted that the token-based approach Section 5.4 provides authorization through token delegation. The strength of this authorization depends on several factors:

1. the uniqueness of tokens: tokens should be constructed in a way that minimize the possibilities for collisions;

2. validity of tokens: applications/users should not re-use tokens; and

3. secrecy of tokens: if tokens are compromised to unauthorized entities, access control for the associated resources cannot be provided.

Depending on the specific application, DECADE can be used to access confidential information. Hence DECADE implementations SHOULD provide a secure transport mode that allows for encryption.

10. IANA Considerations

This document does not have any IANA considerations.

11. Acknowledgments

We thank the following people for their contributions to this document:

David Bryan

Yingjie Gu

David McDysan

Borje Ohlman

Haibin Song

Martin Stiemerling

Richard Woundy

Ning Zong

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

12.2. Informative References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3744] Clemm, G., Reschke, J., Sedlar, E., and J. Whitehead, "Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol", RFC 3744, May 2004.

[RFC4331] Korver, B. and L. Dusseault, "Quota and Size Properties fo r Distributed Authoring and Versioning (DAV) Collections", RFC 4331, February 2006.

[RFC4709] Reschke, J., "Mounting Web Distributed Authoring and Versioning (WebDAV) Servers", RFC 4709, October 2006.

[RFC4918] Dusseault, L., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

[RFC6392] Alimi, R., Rahman, A., and Y. Yang, "A Survey of In-Network Storage Systems", RFC 6392, October 2011.

[I-D.ietf-decade-problem-statement] Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", draft-ietf-decade-problem-statement-04 (work in progress), October 2011.

[I-D.ietf-decade-reqs] Yingjie, G., Bryan, D., Yang, Y., and R. Alimi, "DECADE Requirements", draft-ietf-decade-reqs-04 (work in progress), September 2011.

[GoogleStorageDevGuide] "Google Storage Developer Guide", <http://code.google.com/apis/storage/docs/developer-guide.html>.

[GoogleFileSystem] Ghemawat, S., Gobioff, H., and S. Leung, "The Google File System", SOSP 2003, October 2003.

[CDMI] "CDMI", <http://www.snia.org/cdmi>.

Appendix A. Appendix: Evaluation of Some Candidate Existing Protocols for DECADE DRP and SDT

In this section we evaluate how well the abstract protocol interactions specified in Section 6 for DECADE DRP and SDT can be

fulfilled by existing protocols such as HTTP, WEBDAV, and CDMI.

A.1. HTTP

HTTP [RFC2616] is a key protocol for the Internet in general and especially for the World Wide Web. HTTP is a request-response protocol. A typical transaction involves a client (e.g. web browser) requesting content (resources) from a web server. Another example is when a client stores or deletes content from a server.

A.1.1. HTTP Support for DECADE Resource Protocol Primitives

DRP provides configuration of access control and resource sharing policies on DECADE servers.

A.1.1.1. Access Control Primitives

Access control requires mechanisms for defining the access policies for the server, and then checking the authorization of a user before it stores or retrieves content. HTTP supports a rudimentary access control via "HTTP Secure" (HTTPS). HTTPS is a combination of HTTP with SSL/TLS. The main use of HTTPS is to authenticate the server and encrypt all traffic between the client and the server. There is also a mode to support client authentication though this is less frequently used.

A.1.1.2. Communication Resource Controls Primitives

Communication resources include bandwidth (upload/download) and number of simultaneous connected clients (connections). HTTP supports bandwidth control indirectly through "persistent" HTTP connections. Persistent HTTP connections allows a client to keep open the underlying TCP connection to the server to allow streaming and pipelining (multiple simultaneous requests for a given client).

HTTP does not define protocol operation to allow limiting the communication resources to a client. However servers typically perform this function via implementation algorithms.

A.1.1.3. Storage Resource Control Primitives

Storage resources include amount of memory and lifetime of storage. HTTP does not allow direct control of storage at the server end point. However HTTP supports caching at intermediate points such as a web proxy. For this purpose, HTTP defines cache control mechanisms that define how long and in what situations the intermediate point may store and use the content.

A.1.2. HTTP Support for DECADE Standard Data Transport Protocol Primitives

SDT is used to write objects and read (download) objects from a DECADE server. The object can be either a self-contained object such as a multimedia file or a chunk from a P2P system.

A.1.2.1. Writing Primitives

Writing involves uploading objects to the server. HTTP supports two methods of writing called PUT and POST. In HTTP the object is called a resource and is identified by a URI. PUT uploads a resource to a specific location on the server. POST, on the other hand, submits the object to the server and the server decides whether to update an existing resource or to create a new resource.

For DECADE, the choice of whether to use PUT or POST will be influenced by which entity is responsible for the naming. If the client performs the naming, then PUT is appropriate. If the server performs the naming, then POST should be used (to allow the server to define the URI).

A.1.2.2. Downloading Primitives

Downloading involves fetching of an object from the server. HTTP supports downloading through the GET and HEAD methods. GET fetches a specific resource as identified by the URL. HEAD is similar but only fetches the metadata ("header") associated with the resource but not the resource itself.

A.1.3. Traffic De-duplication Primitives

To challenge a remote entity for an object, the DECADE server should provide a seed number, which is generated by the server randomly, and ask the remote entity to return a hash calculated from the seed number and the content of the object. The server may also specify the hash function which the remote entity should use. HTTP supports the challenge message through the GET methods. The message type ("challenge"), the seed number and the hash function name are put in URL. In the reply, the hash is sent in an ETAG header.

A.1.4. Other Operations

HTTP supports deleting of content on the server through the DELETE method.

A.1.5. Conclusions

HTTP can provide a rudimentary DRP and SDT for some aspects of DECADE, but will not be able to satisfy all the DECADE requirements. For example, HTTP does not provide a complete access control mechanism, nor does it support storage resource controls at the end point server.

It is possible, however, to envision combining HTTP with a custom suite of other protocols to fulfill most of the DECADE requirements for DRP and SDT. For example, Google Storage for Developers is built using HTTP (with extensive proprietary extensions such as custom HTTP headers). Google Storage also uses OAUTH 2.0 (for access control) in combination with HTTP [GoogleStorageDevGuide].

A.2. WEBDAV

WebDAV [RFC4918] is a protocol for enhanced Web content creation and management. It was developed as an extension to HTTP Appendix A.1. WebDAV supports traditional operations for reading/writing from storage, as well as more advanced features such as locking and namespace management which are important when multiple users collaborate to author or edit a set of documents. HTTP is a subset of WebDAV functionality. Therefore, all the points noted above in Appendix A.1 apply implicitly to WebDAV as well.

A.2.1. WEBDAV Support for DECADE Resource Protocol Primitives

DRP provides configuration of access control and resource sharing policies on DECADE servers.

A.2.1.1. Access Control Primitives

Access control requires mechanisms for defining the access policies for the server, and then checking the authorization of a user before it stores or retrieves content. WebDAV has an Access Control Protocol defined in [RFC3744].

The goal of WebDAV access control is to provide an interoperable mechanism for handling discretionary access control for content and metadata managed by WebDAV servers. WebDAV defines an Access Control List (ACL) per resource. An ACL contains a set of Access Control Entries (ACEs), where each ACE specifies a principal (i.e. user or group of users) and a set of privileges that are granted to that principal. When a principal tries to perform an operation on that resource, the server evaluates the ACEs in the ACL to determine if the principal has permission for that operation.

WebDAV also requires that an authentication mechanism be available for the server to validate the identity of a principal. As a minimum, all WebDAV compliant implementations are required to support HTTP Digest Authentication.

A.2.1.2. Communication Resource Controls Primitives

Communications resources include bandwidth (upload/download) and number of simultaneous connected clients (connections). WebDAV supports communication resource control as described in Appendix A.1.1.2.

A.2.1.3. Storage Resource Control Primitives

Storage resources include amount of memory and lifetime of storage. WebDAV supports the concept of properties (which are metadata for a resource). A property is either "live" or "dead". Live properties include cases where a) the value of a property is protected and maintained by the server, and b) the value of the property is maintained by the client, but the server performs syntax checking on submitted values. A dead property has its syntax and semantics enforced by the client; the server merely records the value of the property.

WebDAV supports a list of standardized properties [RFC4918] that are useful for storage resource control. These include the self-explanatory "creationdate", and "getcontentlength" properties. There is also an operation called PROPFIND to retrieve all the properties defined for the requested URI.

WebDAV also has a Quota and Size Properties mechanism defined in [RFC4331] that can be used for storage control. Specifically, two key properties are defined per resource: "quota-available-bytes" and "quota-used-bytes".

WebDAV does not define protocol operation for storage resource control. However servers typically perform this function via implementation algorithms in conjunction with the storage related properties discussed above.

A.2.2. WebDAV Support for DECADE Standard Transport Protocol Primitives

SDT is used to write objects and read (download) objects from a DECADE server. The object can be either a self-contained object such as a multimedia file or a chunk from a P2P system.

A.2.2.1. Writing Primitives

Writing involves uploading objects to the server. WebDAV supports PUT and POST as described in Appendix A.1.2.1. WebDAV LOCK/UNLOCK functionality is not needed as DECADE assumes immutable data objects. Therefore, resources cannot be edited and so do not need to be locked. This approach should help to greatly simplify DECADE implementations as the LOCK/UNLOCK functionality is quite complex.

A.2.2.2. Downloading Primitives

Downloading involves fetching of an object from the server. WebDAV supports GET and HEAD as described in Appendix A.1.2.2. WebDAV LOCK/ UNLOCK functionality is not needed as DECADE assumes immutable data objects.

A.2.3. Other Operations

WebDAV supports DELETE as described in Appendix A.1.4. In addition WebDAV supports COPY and MOVE methods. The COPY operation creates a duplicate of the source resource identified by the Request-URI, in the destination resource identified by the URI in the Destination header.

The MOVE operation on a resource is the logical equivalent of a COPY, followed by consistency maintenance processing, followed by a delete of the source, where all three actions are performed in a single operation. The consistency maintenance step allows the server to perform updates caused by the move, such as updating all URLs, other than the Request-URI that identifies the source resource, to point to the new destination resource.

WebDAV also supports the concept of "collections" of resources to support joint operations on related objects (e.g. file system directories) within a server’s namespace. For example, GET and HEAD may be done on a single resource (as in HTTP) or on a collection. The MKCOL operation is used to create a new collection. DECADE may find the concept of collections to be useful if there is a need to support directory like structures in DECADE.

WebDAV servers can be interfaced from an HTML-based user interface in a web browser. However, it is frequently desirable to be able to switch from an HTML-based view to a presentation provided by a native WebDAV client, directly supporting WebDAV features. The method to perform this in a platform-neutral mechanism is specified in the WebDAV protocol for "mounting WebDAV servers" [RFC4709]. This type of feature may also be attractive for DECADE clients.

A.2.4. Conclusions

WebDAV has a rich array of features that can provide a good base for DRP and SDT for DECADE. An initial analysis finds that the following WebDAV features will be useful for DECADE:

- access control
- properties (and PROPFIND operation)
- COPY/MOVE operations
- collections
- mounting WebDAV servers

It is recommended that the following WebDAV features NOT be used for DECADE:

- LOCK/UNLOCK

Finally, some extensions to WebDAV may still be required to meet all DECADE requirements. For example, defining a new WebDAV "time-to-live" property may be useful for DECADE. Further analysis is required to fully define the potential extensions to WebDAV to meet all DECADE requirements.

A.3. CDMI

The Cloud Data Management Interface (CDMI) specification defines a functional interface through which applications can store and manage data objects in a cloud storage environment. The CDMI interface for reading/writing data is based on standard HTTP requests, with CDMI-specific encodings using JavaScript Object Notation (JSON). CDMI is specified by the Storage Networking Industry Association (SNIA) [CDMI].

A.3.1. CDMI Support for DECADE Resource Protocol Primitives

DRP provides configuration of access control and resource sharing policies on DECADE servers.

A.3.1.1. Access Control Primitives

Access control includes mechanisms for defining the access policies for the server, and then checking the authorization of a user before it stores or retrieves content. CDMI defines an Access Control List (ACL) per data object, and thus supports access control (read and/or

write) at the data object granularity. An ACL contains a set of Access Control Entries (ACEs), where each ACE specifies a principal (i.e. user or group of users) and a set of privileges that are granted to that principal.

CDMI requires that an HTTP authentication mechanism be available for the server to validate the identity of a principal (client). Specifically, CDMI requires that either HTTP Basic Authentication or HTTP Digest Authentication be supported. CDMI recommends that HTTP over TLS (HTTPS) is supported to encrypt the data sent over the network.

A.3.1.2. Communication Resource Controls Primitives

Communication resources include bandwidth (upload/download) and number of simultaneous connected clients (connections). CDMI supports two key data attributes which provide control over the communication resources to a client: "cdmi_max_throughput" and "cdmi_max_latency". These attributes are defined in the metadata for data objects and indicate the desired bandwidth or delay for transmission of the data object from the cloud server to the client.

A.3.1.3. Storage Resource Control Primitives

Storage resources include amount of quantity and lifetime of storage. CDMI defines metadata for individual data objects and general storage system configuration which can be used for storage resource control. In particular, CDMI defines the following metadata fields:

- cdmi_data_redundancy: desired number of copies to be maintained,

- cdmi_geographic_placement: region where object is permitted to be stored,

- cdmi_retention_period: time interval object is to be retained, and

- cdmi_retention_autodelete: whether object should be auto deleted after retention period.

A.3.2. CDMI Support for DECADE Standard Transport Protocol Primitives

SDT is used to write objects and read (download) objects from a DECADE server. The object can be either a self-contained object such as a multimedia file or a chunk from a P2P system.

A.3.2.1. Writing Primitives

Writing involves uploading objects to the server. CDMI supports standard HTTP methods for PUT and POST as described in Appendix A.1.2.1.

A.3.2.2. Downloading Primitives

Downloading involves fetching of an object from the server. CDMI supports the standard HTTP GET method as described in Appendix A.1.2.2.

A.3.3. Other Operations

CDMI supports DELETE as described in Appendix A.1.4. CDMI also supports COPY and MOVE operations.

CDMI supports the concept of containers of data objects to support joint operations on related objects. For example, GET may be done on a single data object or on an entire container.

CDMI supports a global naming scheme. Every object stored within a CDMI system will have a globally unique object string identifier (ObjectID) assigned at creation time.

A.3.4. Conclusions

CDMI has a rich array of features that can provide a good base for DRP and SDT for DECADE. An initial analysis finds that the following CDMI features may be useful for DECADE:

- access control
- storage resource control
- communication resource control
- COPY/MOVE operations
- data containers
- naming scheme

Appendix B. In-Network Storage Components Mapped to DECADE Architecture

In this section we evaluate how the basic components of an in-network storage system identified in Section 3 of [RFC6392] map into the

DECADE architecture.

It is important to note that complex and/or application-specific behavior is delegated to applications instead of tuning the storage system wherever possible.

B.1. Data Access Interface

Users can read and write objects of arbitrary size through the DECADE Client's Data Controller, making use of a standard data transport.

B.2. Data Management Operations

Users can move or delete previously stored objects via the DECADE Client's Data Controller, making use of a standard data transport.

B.3. Data Search Capability

Users can enumerate or search contents of DECADE servers to find objects matching desired criteria through services provided by the Content Distribution Application (e.g., buffer-map exchanges, a DHT, or peer-exchange). In doing so, End-Points may consult their local Data Index in the DECADE Client's Data Controller.

B.4. Access Control Authorization

All methods of access control are supported: public-unrestricted, public-restricted and private. Access Control Policies are generated by a Content Distribution Application and provided to the DECADE Client's Resource Controller. The DECADE Server is responsible for implementing the access control checks.

B.5. Resource Control Interface

Users can manage the resources (e.g. bandwidth) on the DECADE server that can be used by other Application End-Points. Resource Sharing Policies are generated by a Content Distribution Application and provided to the DECADE Client's Resource Controller. The DECADE Server is responsible for implementing the resource sharing policies.

B.6. Discovery Mechanism

The particular protocol used for discovery is outside the scope of this document. However, options and considerations have been discussed in Section 5.5.

B.7. Storage Mode

DECADE Servers provide an object-based storage mode. Immutable data objects may be stored at a DECADE server. Applications may consider existing blocks as DECADE data objects, or they may adjust block sizes before storing in a DECADE server.

Authors' Addresses

Richard Alimi
Google

Email: ralimi@google.com

Y. Richard Yang
Yale University

Email: yry@cs.yale.edu

Akbar Rahman
InterDigital Communications, LLC

Email: akbar.rahman@interdigital.com

Dirk Kutscher
NEC

Email: dirk.kutscher@neclab.eu

Hongqiang Liu
Yale University

Email: hongqiang.liu@yale.edu

DECADE
Internet-Draft
Intended status: Informational
Expires: August 11, 2013

R. Alimi
Google
A. Rahman
InterDigital Communications, LLC
D. Kutscher
NEC
Y. Yang
Yale University
H. Song
K. Pentikousis
Huawei

February 7, 2013

DECADE Protocol
draft-ietf-decade-arch-10

Abstract

Content Distribution Applications (e.g., P2P applications) are widely used on the Internet and make up a large portion of the traffic in many networks. One technique to improve the network efficiency of these applications is to introduce storage capabilities within the networks; this is the capability provided by a DECADE (DECoupled Application Data Enroute) compatible system. This document presents an architecture, discusses the underlying principles, and identifies key functionalities in the architecture for introducing a DECADE in-network storage system. In addition, some examples are given to illustrate these concepts.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 11, 2013.

Table of Contents

1. Introduction 4
2. Terminology 4
3. Protocol Flow 4
3.1. Overview 4
3.2. An Example 5
4. Architectural Principles 6
4.1. Decoupled Control/Metadata and Data Planes 6
4.2. Immutable Data Objects 7
4.3. Data Objects With Identifiers 8
4.4. Explicit Control 9
4.5. Resource and Data Access Control through Delegation . . . 10
5. System Components 11
5.1. Content Distribution Application 11
5.2. DECADE Server 13
5.3. Data Sequencing and Naming 15
5.4. Token-based Authorization and Resource Control 16
5.5. Discovery 17
6. DECADE Protocols 18
6.1. DECADE Naming 18
6.2. DECADE Resource Protocol (DRP) 19
6.3. Standard Data Transfer (SDT) Protocol 23
6.4. Server-to-Server Protocols 24
7. Security Considerations 25
7.1. Threat: System Denial of Service Attacks 25
7.2. Threat: Protocol Security 26
8. IANA Considerations 27
9. Acknowledgments 27
10. References 28
10.1. Normative References 28
10.2. Informative References 28
Appendix A. In-Network Storage Components Mapped to DECADE Architecture 29
A.1. Data Access Interface 29
A.2. Data Management Operations 29
A.3. Data Search Capability 29
A.4. Access Control Authorization 29
A.5. Resource Control Interface 30
A.6. Discovery Mechanism 30
A.7. Storage Mode 30
Appendix B. Hisotry 30
Authors' Addresses 30

1. Introduction

Content Distribution Applications, such as Peer-to-Peer (P2P) applications, are widely used on the Internet to distribute data, and they contribute a large portion of the traffic in many networks. The architecture described in this document enables such applications to leverage in-network storage to achieve more efficient content distribution (i.e. DECADE system). Specifically, in many subscriber networks, it can be expensive to upgrade network equipment in the "last-mile", because it can involve replacing equipment and upgrading wiring at individual homes, businesses, and devices such as DSLAMs (Digital Subscriber Line Access Multiplexers) and CMTSs (Cable Modem Termination Systems) in remote locations. Therefore, it may be cheaper to upgrade the core infrastructure, which involves fewer components that are shared by many subscribers. See [RFC6646] for a more complete discussion of the problem domain and general discussions of the capabilities to be provided by a DECADE system.

This document presents an architecture for providing in-network storage that can be integrated into Content Distribution Applications. The primary focus is P2P-based content distribution, but the architecture may be useful to other applications with similar characteristics and requirements. See [I-D.ietf-decade-reqs] for a definition of the target applications as well as the requirements for a DECADE system.

The approach of this document is to define the core functionalities and protocol functions that are needed to support a DECADE system. The specific protocols are not selected or designed in this document. Some illustrative examples are given to help the reader understand certain concepts. These examples are purely informational and are not meant to constrain future protocol design or selection.

2. Terminology

This document assumes readers are familiar with the terms and concepts that are used in [RFC6646] and [I-D.ietf-decade-reqs].

3. Protocol Flow

3.1. Overview

Following[I-D.ietf-decade-reqs], the architecture consists of two protocols: the DECADE Resource Protocol (DRP) that is responsible for communication of access control and resource scheduling policies from a client to a server, as well as between servers; and Standard Data

Transfer (SDT) protocol(s) that will be used to transfer data objects to and from a server. We show the protocol components figure below:

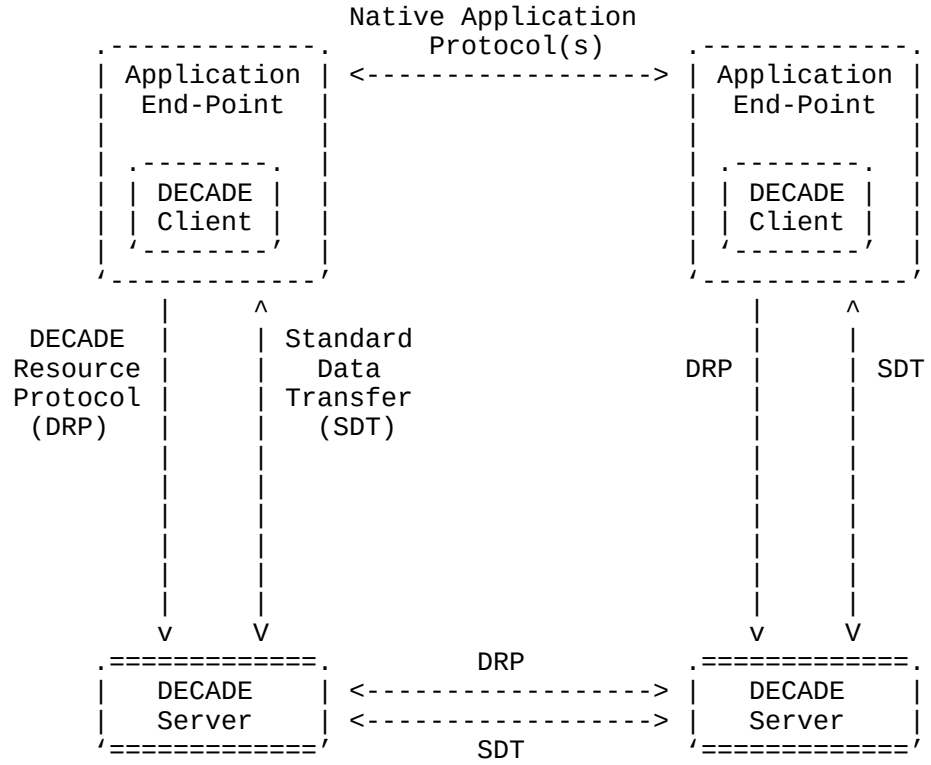


Figure 1: Generic Protocol Flow

3.2. An Example

This section provides an example showing the steps in the architecture for a data transfer scenario involving an in-network storage system. We assume that Application End-Point B (the receiver) is requesting a data object from Application End-Point A (the sender). Let S(A) denote the DECADE storage server to which A has access. There are multiple usage scenarios (by choice of the Content Distribution Application). For simplicity of introduction, we design this example to use only a single DECADE server.

The steps of the example are illustrated in Figure 2. First, B requests a data object from A using their native application protocol (see Section 5.1.2). Next, A uses the DRP to obtain a token. There are multiple ways for A to obtain the token: compute locally, or request from its DECADE storage server, S(A). See Section 6.2.2 for

details. A then provides the token to B (again, using their native application protocol). Finally, B provides the token to S(A) via DRP, and requests and downloads the data object via a SDT.

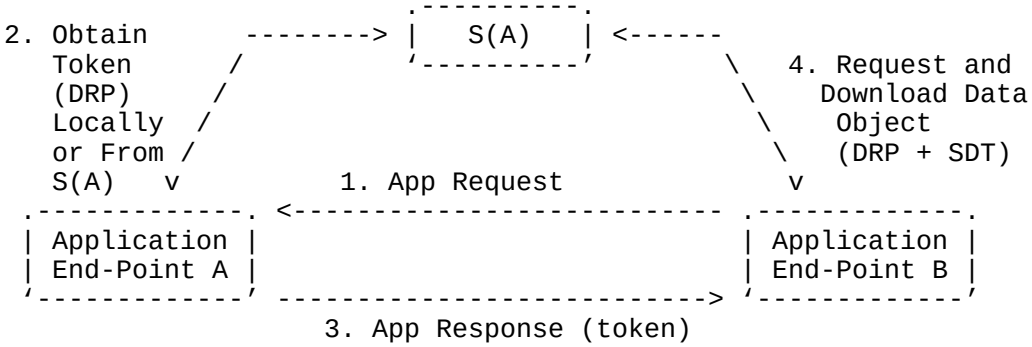


Figure 2: Download from Storage Server

4. Architectural Principles

We identify the following key principles that will be followed in any DECADE system:

4.1. Decoupled Control/Metadata and Data Planes

A DECADE system SHOULD be able to support multiple Content Distribution Applications. A complete Content Distribution Application implements a set of "control plane" functions including content search, indexing and collection, access control, replication, request routing, and QoS scheduling. Different Content Distribution Applications will have unique considerations designing the control plane functions:

- Metadata Management Scheme: Traditional file systems provide a standard metadata abstraction: a recursive structure of directories to offer namespace management; each file is an opaque byte stream. Content Distribution Applications may use different metadata management schemes. For example, one application might use a sequence of blocks (e.g., for file sharing), while another application might use a sequence of frames (with different sizes) indexed by time.

- Resource Scheduling Algorithms: A major advantage of many successful P2P systems is their substantial expertise in achieving highly efficient utilization of peer and infrastructural

resources. For instance, many streaming P2P systems have their specific algorithms in constructing topologies to achieve low-latency, high-bandwidth streaming. They continuously fine-tune such algorithms.

Given the diversity of control plane functions, a DECADE system SHOULD allow as much flexibility as possible to the control plane to implement specific policies. This conforms to the end-to-end systems principle and allows innovation and satisfaction of specific performance goals.

Decoupling control plane and data plane is not new. For example, OpenFlow [OpenFlow] is an implementation of this principle for Internet routing, where the computation of the forwarding table and the application of the forwarding table are separated. Google File System [GoogleFileSystem] applies the principle to file system design, by utilizing the Master to handle the meta-data management, and the Chunk servers to handle the data plane functions (i.e., read and write of chunks of data). NFSv4.1's pNFS extension [RFC5661] also implements this principle.

4.2. Immutable Data Objects

A property of bulk content to be broadly distributed is that they typically are immutable -- once content is generated, it is typically not modified. It is not common that bulk content such as video frames and images need to be modified after distribution.

Focusing on immutable data in the data plane can substantially simplify the data plane design, since consistency requirements can be relaxed. It also simplifies reuse of data and implementation of de-duplication.

Depending on its specific requirements, an application may store immutable data objects in DECADE servers such that each data object is completely self-contained (e.g., a complete, independently decodable video segment). An application may also divide data into data objects that require application level assembly. Many Content Distribution Applications divide bulk content into data objects for multiple reasons, including (1) fetching different data objects from different sources in parallel; and (2) faster recovery and verification: individual data objects might be recovered and verified. Typically, applications use a data object size larger than a single packet in order to reduce control overhead.

A DECADE system SHOULD be agnostic to the nature of the data objects and SHOULD NOT specify a fixed size for them. A protocol specification based on this architecture MAY prescribe requirements

on minimum and maximum sizes by compliant implementations.

Immutable data objects can still be deleted. Applications may support modification of existing data stored at a DECADE server through a combination of storing new data objects and deleting existing data objects. For example, a meta-data management function of the control plane might associate a name with a sequence of immutable data objects. If one of the data objects is modified, the meta-data management function changes the mapping of the name to a new sequence of immutable data objects.

Throughout this document, all data objects are assumed to be immutable.

4.3. Data Objects With Identifiers

An object that is stored in a DECADE storage server SHALL be accessed by Content Consumers via a data object identifier.

A Content Consumer may be able to access more than one storage server. A data object that is replicated across different storage servers managed by a DECADE Storage Provider MAY still be accessed by a single identifier.

Since data objects are immutable, it SHALL be possible to support persistent identifiers for data objects.

Data object identifiers for data objects SHOULD be created by Content Providers that upload the objects to servers. We refer to a scheme for the assignment/derivation of the data object identifier to a data object depends as the data object naming scheme. The details of data naming schemes will be provided by future DECADE protocol/naming specifications. This document describes naming schemes on a semantic level and specific SDTs and DRPs SHOULD use specific representations.

In particular, for some applications it is important that clients and servers SHOULD be able to validate the name-object binding for a data object, i.e., by verifying that a received object really corresponds to the name (identifier) that was used for requesting it (or that was provided by a sender). Data object identifiers can support name-object binding validation by providing message digests or so-called self-certifying naming information -- if a specific application has this requirement.

A DECADE naming scheme follows the following general requirements:

- Different name-object binding validation mechanisms MAY be supported;

- Content Distribution Applications will decide what mechanism to use, or to not provide name-object validation (e.g., if authenticity and integrity can by ascertained by alternative means);

- Applications MAY be able to construct unique names (with high probability) without requiring a registry or other forms of coordination; and

- Names MAY be self-describing so that a receiving entity (Content Consumer) knows what hash function (for example) to use for validating name-object binding.

Some Content Distribution Applications will derive the name of a data object from the hash over the data object, which is made possible by the fact that DECADE objects are immutable. But there may be other applications such as live streaming where object names will not based on hashes but rather on an enumeration scheme. The naming scheme will also enable those applications to construct unique names.

In order to enable the uniqueness, flexibility and self-describing properties, the naming scheme SHOULD provide the following name elements:

- A "type" field that indicates the name-object validation function type (for example, "sha-256");

- Cryptographic data (such as an object hash) that corresponds to the type information; and

The naming scheme MAY additionally provide the following name elements:

- Application or publisher information.

The specific format of the name (e.g., encoding, hash algorithms, etc) is out of scope of this document, and is left for protocol specification.

4.4. Explicit Control

To support the functions of an application’s control plane, applications SHOULD be able to know and coordinate which data is stored at particular servers. Thus, in contrast with traditional caches, applications are given explicit control over the placement (selection of a DECADE server), deletion (or expiration policy), and access control for stored data.

Consider deletion/expiration policy as a simple example. An application might require that a server stores data objects for a relatively short period of time (e.g., for live-streaming data). Another application might need to store data objects for a longer duration (e.g., for video-on-demand).

4.5. Resource and Data Access Control through Delegation

A DECADE system will provide a shared infrastructure to be used by multiple Content Consumers and Content Providers spanning multiple Content Distribution Applications. Thus, it needs to provide both resource and data access control.

4.5.1. Resource Allocation

There are two primary interacting entities in a DECADE system. First, Storage Providers SHOULD coordinate where storage servers are provisioned and their total available resources Section 6.2.1. Second, Applications will coordinate data transfers amongst available servers and between servers and clients. A form of isolation is required to enable concurrently-running Applications to each explicitly manage its own data objects and share of resources at the available servers.

The Storage Provider SHOULD delegate the management of the resources on a server to Content Providers. This means that Content Providers are able to explicitly and independently manage their own shares of resources on a server.

4.5.2. User Delegations

Storage Providers will have the ability to explicitly manage the entities allowed to utilize the resources at a DECADE server. This capability is needed for reasons such as capacity-planning and legal considerations in certain deployment scenarios.

The server SHOULD grant a share of the resources to a Content Provider or Content Consumer. The client can in turn share the granted resources amongst its multiple applications. The share of resources granted by a server is called a User Delegation.

As a simple example, a DECADE server operated by an ISP might be configured to grant each ISP Subscriber 1.5 Mbit/s of bandwidth. The ISP Subscriber might in turn divide this share of resources amongst a video streaming application and file-sharing application which are running concurrently.

5. System Components

The primary focus of this document is the architectural principles and the system components that implement them. While certain system components might differ amongst implementations, the document details the major components and their overall roles in the architecture.

To keep the scope narrow, we only discuss the primary components related to protocol development. Particular deployments will require additional components (e.g., monitoring and accounting at a server), but they are intentionally omitted from this document.

5.1. Content Distribution Application

Content Distribution Applications have many functional components. For example, many P2P applications have components and algorithms to manage overlay topology management, rate allocation, piece selection, etc. In this document, we focus on the components directly employed to support a DECADE system.

Figure 3 illustrates the components discussed in this section from the perspective of a single Application End-Point.

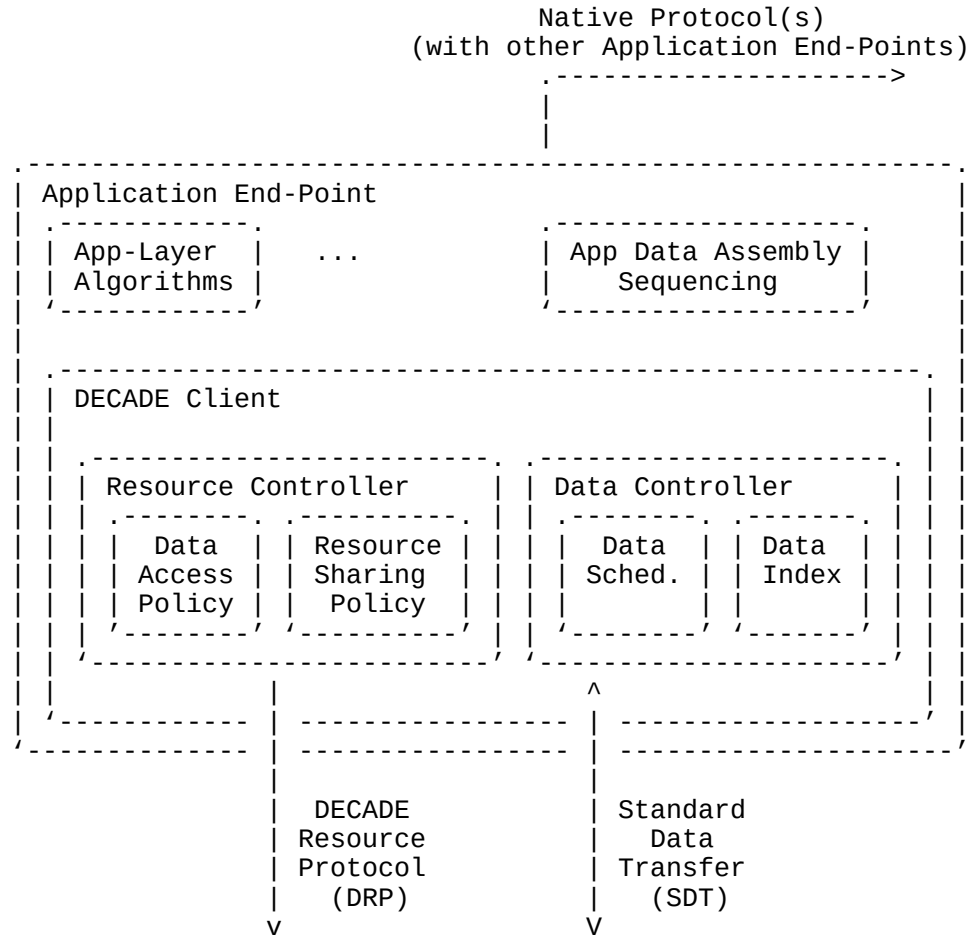


Figure 3: Application Components

5.1.1. Data Assembly

A DECADE system is geared towards supporting applications that can distribute content using data objects. To accomplish this, applications can include a component responsible for creating the individual data objects before distribution and then re-assembling data objects at the Content Consumer. We call this component the Application Data Assembly.

In producing and assembling the data objects, two important considerations are sequencing and naming. A DECADE system assumes that applications implement this functionality themselves. See Section 6.1 for further discussion.

5.1.2. Native Application Protocols

In addition to the DECADE DRP/SDT, applications can also support existing native application protocols (e.g., P2P control and data transfer protocols).

5.1.3. DECADE Client

The client provides the local support to an application, and can be implemented standalone, embedded into the application, or integrated in other entities such as network devices themselves.

5.1.3.1. Resource Controller

Applications may have different Resource Sharing Policies and Data Access Policies to control their resource and data in DECADE servers. These policies may be existing policies of applications or custom policies. The specific implementation is decided by the application.

5.1.3.2. Data Controller

A DECADE system decouples the control and the data transfer of applications. A Data Scheduling component schedules data transfers according to network conditions, available servers, and/or available server resources. The Data Index indicates data available at remote servers. The Data Index (or a subset of it) can be advertised to other clients. A common use case for this is to provide the ability to locate data amongst distributed Application End-Points (i.e., a data search mechanism such as a Distributed Hash Table).

5.2. DECADE Server

Figure 4 illustrates the components discussed in a DECADE server. A server is not necessarily a single physical machine, it can also be implemented as a cluster of machines.

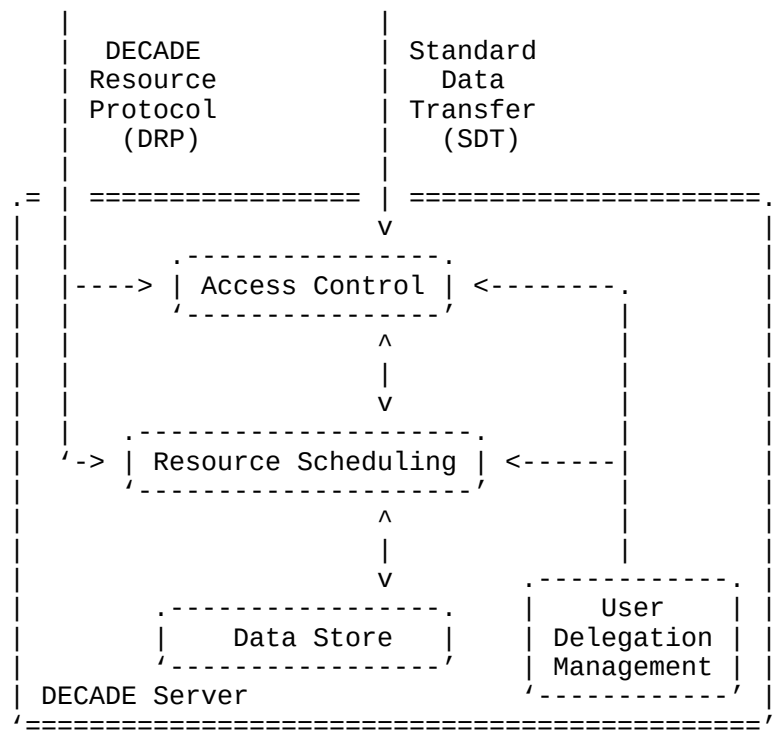


Figure 4: DECADE Server Components

5.2.1. Access Control

A client SHALL be able to access its own data or other client's data (provided sufficient authorization) in DECADE servers. Clients MAY also authorize other clients to store data. If an access is authorized by a client, the server SHOULD provide access. Even if a request is authorized, it MAY still fail to complete due to insufficient resources at the server.

5.2.2. Resource Scheduling

Applications will apply resource sharing policies or use a custom policy. Servers perform resource scheduling according to the resource sharing policies indicated by clients as well as configured User Delegations.

5.2.3. Data Store

Data from applications will be stored at a DECADE server. Data may be deleted from storage either explicitly or automatically (e.g.,

after a TTL expiration).

5.3. Data Sequencing and Naming

The DECADE naming scheme implies no sequencing or grouping of objects, even if this is done at the application layer.

5.3.1. Application Usage Example

To illustrate these properties, this section presents multiple examples.

5.3.1.1. Application with Fixed-Size Chunks

Similar to the example in Section 5.1.1, consider an Application in which each individual application-layer segment of data is called a "chunk" and has a name of the form: "CONTENT_ID:SEQUENCE_NUMBER". Furthermore, assume that the application's native protocol uses chunks of size 16 KiB.

Now, assume that this application wishes to store data in DECADE servers in data objects of size 64 KiB. To accomplish this, it can map a sequence of 4 chunks into a single data object, as shown in Figure 5.

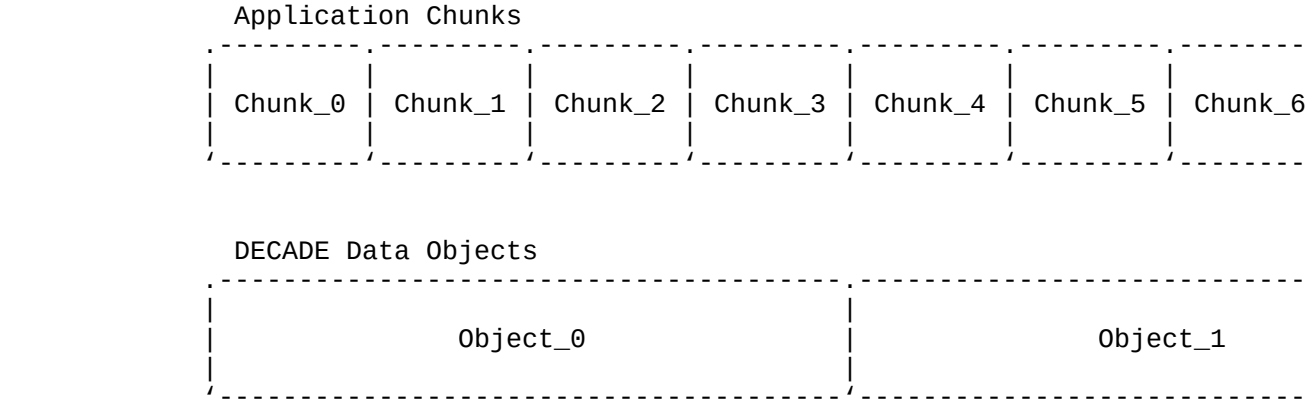


Figure 5: Mapping Application Chunks to DECADE Data Objects

In this example, the Application maintains a logical mapping that is able to determine the name of a DECADE data object given the chunks contained within that data object. The name may be learned from either the original Content Provider, another End-Point with which the Application is communicating, etc. As long as the data contained within each sequence of chunks is globally unique, the corresponding

data objects have globally unique names.

5.3.1.2. Application with Continuous Streaming Data

Consider an Application whose native protocol retrieves a continuous data stream (e.g., an MPEG2 stream) instead of downloading and redistributing chunks of data. Such an application could segment the continuous data stream to produce either fixed-sized or variable-sized data objects.

Figure 6 shows how a video streaming application might produce variable-sized data objects such that each data object contains 10 seconds of video data.

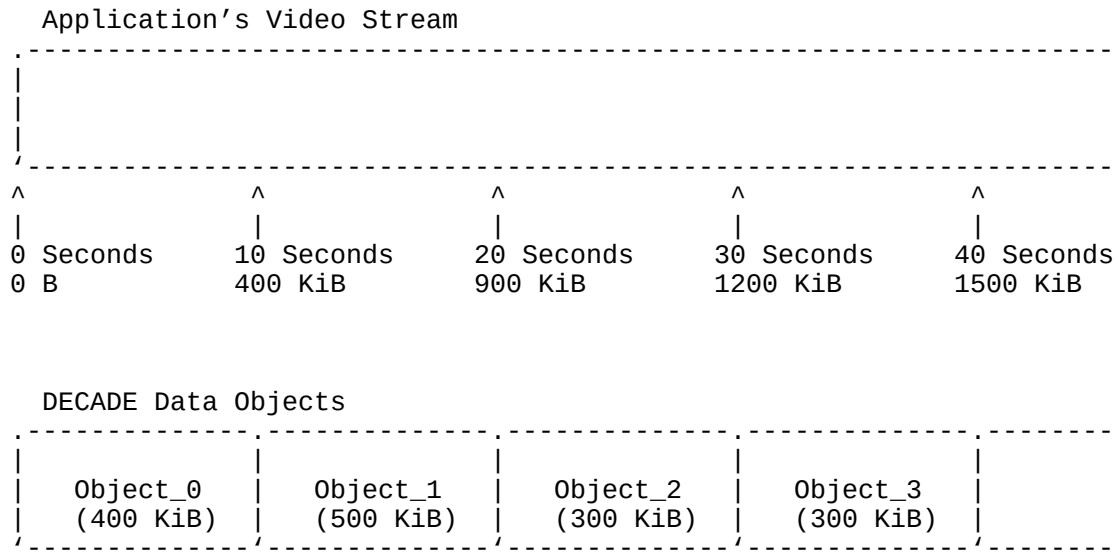


Figure 6: Mapping a Continuous Data Stream to DECADE Data Objects

Similar to the previous example, the Application might maintain a mapping that is able to determine the name of a data object given the time offset of the video chunk.

5.4. Token-based Authorization and Resource Control

A key feature of a DECADE system is that an application endpoint can authorize other application endpoint to store or retrieve data objects from the in-network storage. An OAuth version 2 [RFC6749]based authorization scheme is used to accomplish this. A separate OAuth flow is used for this purpose,

a client authenticates (optional and out of the scope of this document) with the application server or the P2P application peer, and request the trusted by the client, and the token contains particular self contained properties (see Section 6.2.2 for details). The client then use the token when sending requests to the DECADE server. Upon receiving a token, the server validates the signature and the operation being performed.

This is a simple scheme, but has some important advantages over an alternative approach in which a client explicitly manipulates an Access Control List (ACL) associated with each data object. In particular, it has the following advantages when applied to DECADE target applications:

- o Authorization policies are implemented within the Application; an Application explicitly controls when tokens are generated and to whom they are distributed and for how long they will be valid.

- o Fine-grained access and resource control can be applied to data objects; see Section 6.2.2 for the list of restrictions that can be enforced with a token.

- o There is no messaging between a client and server to manipulate data object permissions. This can simplify, in particular, Applications which share data objects with many dynamic peers and need to frequently adjust access control policies attached to data objects.

- o Tokens can provide anonymous access, in which a server does not need to know the identity of each client that accesses it. This enables a client to send tokens to clients belonging to other Storage Providers, and allow them to read or write data objects from the storage of its own Storage Provider.

In addition to clients applying access control policies to data objects, the server MAY be configured to apply additional policies based on user, object, geographic location, etc. A client might thus be denied access even though it possesses a valid token.

There are existing protocols (e.g., OAuth [RFC5849]) that implement similar referral mechanisms using tokens. A protocol specification of this architecture SHOULD endeavor to use existing mechanisms wherever possible.

5.5. Discovery

A DECADE system SHOULD include a discovery mechanism through which clients locate an appropriate server. [I-D.ietf-decade-reqs] details

specific requirements of the discovery mechanism; this section discusses how they relate to other principles outlined in this document.

A discovery mechanism SHOULD allow a client to determine an IP address or some other identifier that can be resolved to locate the server for which the client will be authorized to generate tokens (via DRP). (The discovery mechanism might also result in an error if no such servers can be located.) After discovering one or more servers, a client can distribute load and requests across them (subject to resource limitations and policies of the servers themselves) according to the policies of the Application End-Point in which it is embedded.

The particular protocol used for discovery is out of scope of this document, but any specification SHOULD re-use standard protocols wherever possible.

The discovery mechanism outlined here does not provide the ability to locate arbitrary DECADE servers to which a client might obtain tokens from others. To do so will require application-level knowledge, and it is assumed that this functionality is implemented in the Content Distribution Application.

6. DECADE Protocols

This section presents the DRP and the SDT protocol in terms of abstract protocol interactions that are intended to be mapped to specific protocols. In general, the DRP/SDT functionality between a DECADE client-server are very similar to the DRP/SDT functionality between server-server. Any differences are highlighted below.

DRP will be the protocol used by a DECADE client to configure the resources and authorization used to satisfy requests (reading, writing, and management operations concerning data objects) at a server. SDT will be used to transport data between a client and a server.

6.1. DECADE Naming

A DECADE system SHOULD use the [I-D.farrell-decade-ni] as the recommended and default naming scheme. Other naming schemes that meet the guidelines in Section 4.3 may alternatively be used.

In order to provide a simple and generic interface, the DECADE server will be responsible only for storing and retrieving individual data objects.

The DECADE naming format SHOULD NOT attempt to replace any naming or sequencing of data objects already performed by an Application; instead, the naming is intended to apply only to data objects referenced by DECADE-specific purposes.

An Application using a DECADE client may use a naming and sequencing scheme independent of DECADE names. The DECADE client SHOULD maintain a mapping from its own data objects and their names to the DECADE-specific data objects and names. Furthermore, the DECADE naming scheme implies no sequencing or grouping of objects, even if this is done at the application layer.

6.2. DECADE Resource Protocol (DRP)

DRP will provide configuration of access control and resource sharing policies on DECADE servers. A Content Distribution Application, e.g., a live P2P streaming session, can have permission to manage data at several servers, for instance, servers belonging to different Storage Providers, and DRP allows one instance of such an application, e.g., an Application End-Point, to apply access control and resource sharing policies on each of them.

6.2.1. Controlled Resources

On a single DECADE server, the following resources SHOULD be managed:

- Communication resources in terms of bandwidth (upload/download) and also in terms of number of active clients (simultaneous connections).

- Storage resources.

6.2.2. Access and Resource Control Token

As in DECADE system, the resource owner agent is always the same entity or colocated with the authorization server, so we use a separate OAuth 2.0 request and response flow for the access and resource control token.

An OAuth request to access the data objects MUST include the following fields (encoding format is TBD, HTML?):

response_type: REQUIRED. Value MUST be set to "token".

client_id: the client_id indicates either the application that is using the DECADE service or the end user who is using the DECADE service from a DECADE storage service provider. DECADE storage service providers MUST provide the ID distribution and management

function, which is out of the scope of this document.

scope: data object names that are requested.

An OAuth response includes the following information (encoding is TBD, HTML is preferred, are we going to use OAuth Bearer token type as defined in RFC 6750? The concern for bearer token is that it does not associate the token with any client, so that any client can use this token to access the resources. Do we worry about it? The current draft seems explicitly support this behavior.):

- token_type: "Bearer"?
- expires_in: The lifetime in seconds of the access token.
- access_token: a token denotes the following information.
- service URI: the server address or URI which is providing the service;
- Permitted operations (e.g., read, write);
- Permitted objects (e.g., names of data objects that might be read or written);
- Priority: optional. If it is presented, value MUST be set to be either "Urgent", "High", "Normal" or "Low".
- Bandwidth: bandwidth that is given to requested operation, a weight value used in a weighted bandwidth sharing scheme, or a integer in number of bps;
- Amount: data size in number of bytes that might be read or written.
- token_signature: the signature of the access token.

The tokens SHOULD be generated by an entity trusted by both the DECADE client and server at the request of a DECADE client. For example this entity could be the client, a server trusted by the client, or another server managed by a Storage Provider and trusted by the client. It is important for a server to trust the entity generating the tokens since each token may incur a resource cost on the server when used. Likewise, it is important for a client to trust the entity generating the tokens since the tokens grant access to the data stored at the server.

Upon generating a token, a client MAY distribute it to another client

(e.g., via their native application protocol). The receiving client MAY then connect to the server specified in the token and perform any operation permitted by the token. The token SHOULD be sent along with the operation. The server SHOULD validate the token to identify the client that issued it and whether the requested operation is permitted by the contents of the token. If the token is successfully validated, the server SHOULD apply the resource control policies indicated in the token while performing the operation.

Tokens SHOULD include a unique identifier to allow a server to detect when a token is used multiple times and reject the additional usage attempts. Since usage of a token incurs resource costs to a server (e.g., bandwidth and storage) and a Content Provider may have a limited budget (see Section 4.5), the Content Provider should be able to indicate if a token may be used multiple times.

It SHOULD be possible to revoke tokens after they are generated. This could be accomplished by supplying the server the unique identifiers of the tokens which are to be revoked.

6.2.3. Status Information

DRP SHOULD provide a status request service that clients can use to request status information of a server.

6.2.3.1. Status Information on a specific server

Access to such status information SHOULD require client authorization; that is, clients need to be authorized to access the requested status information. This authorization is based on the user delegation concept as described in Section 4.5. The following status information elements SHOULD be obtained:

- List of associated data objects (with properties);
- Resources used/available.

The following information elements MAY additionally be available:

- List of servers to which data objects have been distributed (in a certain time-frame);
- List of clients to which data objects have been distributed (in a certain time-frame).

For the list of servers/clients to which data objects have been distributed to, the server SHOULD be able to decide on time bounds for which this information is stored and specify the corresponding

time frame in the response to such requests. Some of this information may be used for accounting purposes, e.g., the list of clients to which data objects have been distributed.

6.2.3.2. Access information on a specific server

Access information MAY be provided for accounting purposes, for example, when Content Providers are interested in access statistics for resources and/or to perform accounting per user. Again, access to such information requires client authorization and SHOULD based on the delegation concept as described in Section 4.5. The following type of access information elements MAY be requested:

- What data objects have been accessed by whom and for how many times;
- Access tokens that a server as seen for a given data object.

The server SHOULD decide on time bounds for which this information is stored and specify the corresponding time frame in the response to such requests.

6.2.4. Data Object Attributes

Data Objects that are stored on a DECADE server SHOULD have associated attributes (in addition to the object identifier and data object) that relate to the data storage and its management. These attributes may be used by the server (and possibly the underlying storage system) to perform specialized processing or handling for the data object, or to attach related server or storage-layer properties to the data object. These attributes have a scope local to a server. In particular, these attributes SHOULD NOT be applied to a server or client to which a data object is copied.

Depending on authorization, clients SHOULD be permitted to get or set such attributes. This authorization is based on the delegation concept as described in Section 4.5. The architecture does not limit the set of permissible attributes, but rather specifies a set of baseline attributes that SHOULD be supported:

Expiration Time: Time at which the data object can be deleted;

Data Object size: In bytes;

Media type Labelling of type as per [RFC4288];

Access statistics: How often the data object has been accessed (and what tokens have been used).

The data object attributes defined here are distinct from application metadata (see Section 4.1). Application metadata is custom information that an application might wish to associate with a data object to understand its semantic meaning (e.g., whether it is video and/or audio, its playback length in time, or its index in a stream). If an application wishes to store such metadata persistently, it can be stored within data objects themselves.

6.3. Standard Data Transfer (SDT) Protocol

A DECADE server will provide a data access interface, and the SDT will be used to write data objects to a server and to read (download) data objects from a server. Semantically, SDT is a client-server protocol; that is, the server always responds to client requests.

6.3.1. Writing/Uploading Objects

To write a data object, a client first generates the object's name (see Section 6.1), and then uploads the object to a server and supplies the generated name. The name can be used to access (download) the object later; for example, the client can pass the name as a reference to other client that can then refer to the object.

Data objects can be self-contained objects such as multimedia resources, files etc., but also chunks, such as chunks of a P2P distribution protocol that can be part of a containing object or a stream.

The application that originates the data objects generates DECADE object names according to the naming specification in Section 6.1. Clients (as parts of application entities) upload a named object to a server. If supported, a server can verify the integrity and other security properties of uploaded objects.

6.3.2. Downloading Data Objects

A client can request named data objects from a server. In a corresponding request message, a client specifies the object name and a suitable access and resource control token. The server checks the validity of the received token and its associated resource usage-related properties.

If the named data object exists on the server and the token can be validated, the server delivers the requested object in a response

message.

If the data object cannot be delivered the server provides an corresponding status/reason information in a response message.

Specifics regarding error handling, including additional error conditions (e.g., overload), precedence for returned errors and its relation with server policy, are deferred to eventual protocol specification.

6.4. Server-to-Server Protocols

An important feature of a DECADE system is the capability for one server to directly download data objects from another server. This capability allows Applications to directly replicate data objects between servers without requiring end-hosts to use uplink capacity to upload data objects to a different server.

DRP and SDT will support operations directly between servers. Servers are not assumed to trust each other nor are configured to do so. All data operations are performed on behalf of clients via explicit instruction. However, the objects being processed do not necessarily have to originate or terminate at the client (i.e., the data object might be limited to being exchanged between servers even if the instruction is triggered by the client). Clients thus will be able to indicate to a server the following additional parameters:

- Which remote server(s) to access;
- The operation to be performed;
- The Content Provider at the remote server from which to retrieve the data object, or in which the object is to be stored; and
- Credentials indicating access and resource control to perform the operation at the remote server.

Server-to-server support is focused on reading and writing data objects between servers. The data object referred to at the remote server is the same as the original data object requested by the client. Object attributes (see Section 6.2.4) might also be specified in the request to the remote server.

In this way, a server acts as a proxy for a client, and a client can instantiate requests via that proxy. The operations will be performed as if the original requester had its own client co-located with the server.

When a client sends a request to a server with these additional parameters, it is giving the server permission to act (proxy) on its behalf. Thus, it would be prudent for the supplied token to have narrow privileges (e.g., limited to only the necessary data objects) or validity time (e.g., a small expiration time).

In the case of a retrieval operation, the server is to retrieve the data object from the remote server using the specified credentials, and then optionally return the object to a client. In the case of a storage operation, the server is to store the object to the remote server using the specified credentials. The object might optionally be uploaded from the client or might already exist at the proxy server.

7. Security Considerations

In general, the security considerations mentioned in [RFC6646] apply to this document as well.

A DECADE system provides a distributed storage service for content distribution and similar applications. The system consists of servers and clients that use these servers to upload data objects, to request distribution of data objects, and to download data objects. Such a system is employed in an overall application context -- for example in a P2P Content Distribution Application, and it is expected that DECADE clients take part in application-specific communication sessions.

The security considerations here focus on threats related to the DECADE system and its communication services, i.e., the DRP/SDT protocols that have been described in an abstract fashion in this document.

7.1. Threat: System Denial of Service Attacks

A DECADE network might be used to distribute data objects from one client to a set of servers using the server-to-server communication feature that a client can request when uploading an object. Multiple clients uploading many objects at different servers at the same time and requesting server-to-server distribution for them could thus mount massive distributed denial of service (DDOS) attacks, overloading a network of servers.

This threat is addressed by the server’s access control and resource control framework. Servers can require Application End-Points to be authorized to store and to download objects, and Application End-Points can delegate authorization to other Application End-Points

using the token mechanism.

Of course the effective security of this approach depends on the strength of the token mechanism. See below for a discussion of this and related communication security threats.

Denial of Service Attacks against a single server (directing many requests to that server) might still lead to considerable load for processing requests and invalidating tokens. SDT therefore MUST provide a redirection mechanism as described as a requirement in [I-D.ietf-decade-reqs].

7.2. Threat: Protocol Security

7.2.1. Threat: Authorization Mechanisms Compromised

A DECADE system does not require Application End-Points to authenticate in order to access a server for downloading objects, since authorization is not based on End-Point or user identities but on the delegation-based authorization mechanism. Hence, most protocol security threats are related to the authorization scheme.

The security of the token mechanism depends on the strength of the token mechanism and on the secrecy of the tokens. A token can represent authorization to store a certain amount of data, to download certain objects, to download a certain amount of data per time etc. If it is possible for an attacker to guess, construct or simply obtain tokens, the integrity of the data maintained by the servers is compromised.

This is a general security threat that applies to authorization delegation schemes. Specifications of existing delegation schemes such as OAuth [RFC5849] discuss these general threats in detail. We can say that the DRP has to specify appropriate algorithms for token generation. Moreover, authorization tokens should have a limited validity period that should be specified by the application. Token confidentiality should be provided by application protocols that carry tokens, and the SDT and DRP should provide secure (confidential) communication modes.

7.2.2. Threat: Data Object Spoofing

In a DECADE system, an Application End-Point is referring other Application End-Points to servers to download a specified data objects. An attacker could "inject" a faked version of the object into this process, so that the downloading End-Point effectively receives a different object (compared to what the uploading End-Point provided). As result, the downloading End-Point believes that is has

received an object that corresponds to the name it was provided earlier, whereas in fact it is a faked object. Corresponding attacks could be mounted against the application protocol (that is used for referring other End-Points to servers), servers themselves (and their storage sub-systems), and the SDT by which the object is uploaded, distributed and downloaded.

A DECADE systems fundamental mechanism against object spoofing is name-object binding validation, i.e., the ability of a receiver to check whether the name he was provided and that he used to request an object, actually corresponds to the bits he received. As described above, this allows for different forms of name-object binding, for example using hashes of data objects, with different hash functions (different algorithms, different digest lengths). For those application scenarios where hashes of data objects are not applicable (for example live-streaming) other forms of name-object binding can be used (see Section 6.1). This flexibility also addresses cryptographic algorithm evolvability: hash functions might get deprecated, better alternatives might be invented etc., so that applications can choose appropriate mechanisms meeting their security requirements.

DECADE servers MAY perform name-object binding validation on stored objects, but Application End-Points MUST NOT rely on that. In other words, Application End-Points SHOULD perform name-object binding validation on received objects.

8. IANA Considerations

This document does not have any IANA considerations.

9. Acknowledgments

We thank the following people for their contributions to and/or detailed reviews of this document:

Carsten Bormann

David Bryan

Dave Crocker

Yingjie Gu

David Harrington

Hongqiang (Harry) Liu

David McDysan

Borje Ohlman

Konstantinos Pentikousis

Martin Stiemerling

Richard Woundy

Ning Zong

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6646] Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", RFC 6646, July 2012.

[I-D.ietf-decade-reqs] Yingjie, G., Bryan, D., Yang, Y., Zhang, P., and R. Alimi, "DECADE Requirements", draft-ietf-decade-reqs-08 (work in progress), August 2012.

[I-D.farrell-decade-ni] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keraenen, A., and P. Hallam-Baker, "Naming Things with Hashes", draft-farrell-decade-ni-10 (work in progress), August 2012.

10.2. Informative References

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", RFC 4288, December 2005.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

[RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.

[RFC6392] Alimi, R., Rahman, A., and Y. Yang, "A Survey of In-Network Storage Systems", RFC 6392, October 2011.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[OpenFlow] "OpenFlow Organization", <http://www.openflow.org/>.

[GoogleFileSystem] Ghemawat, S., Gobioff, H., and S. Leung, "The Google File System", SOSP 2003, October 2003.

Appendix A. In-Network Storage Components Mapped to DECADE Architecture

In this section we evaluate how the basic components of an in-network storage system identified in Section 3 of [RFC6392] map into a DECADE system.

A.1. Data Access Interface

Clients can read and write objects of arbitrary size through the client's Data Controller, making use of a SDT.

A.2. Data Management Operations

Clients can move or delete previously stored objects via the client's Data Controller, making use of a SDT.

A.3. Data Search Capability

Clients can enumerate or search contents of servers to find objects matching desired criteria through services provided by the Content Distribution Application (e.g., buffer-map exchanges, a DHT, or peer-exchange). In doing so, Application End-Points might consult their local Data Index in the client's Data Controller.

A.4. Access Control Authorization

All methods of access control are supported: public-unrestricted, public-restricted and private. Access Control Policies are generated by a Content Distribution Application and provided to the client's Resource Controller. The server is responsible for implementing the access control checks.

A.5. Resource Control Interface

Clients can manage the resources (e.g., bandwidth) on the DECADE server that can be used by other Application End-Points. Resource Sharing Policies are generated by a Content Distribution Application and provided to the client's Resource Controller. The server is responsible for implementing the resource sharing policies.

A.6. Discovery Mechanism

The particular protocol used for discovery is outside the scope of this document. However, options and considerations have been discussed in Section 5.5.

A.7. Storage Mode

Servers provide an object-based storage mode. Immutable data objects might be stored at a server. Applications might consider existing blocks as data objects, or they might adjust block sizes before storing in a server.

Appendix B. Hisotry

To RFC Editor: This section is informational for you. Please remove this section before publication.

Since version 10, this document was modified based on the previous DECADE WG architecture document , and was extended to be a protocol specification. It addresses the comments from the WG and the responsible ADs (David Harrington and then Martin Stiemerling). The authors now request to publish this document through the independent stream and get the support of Martin.

Authors' Addresses

Richard Alimi
Google

Email: ralimi@google.com

Akbar Rahman
InterDigital Communications, LLC

Email: akbar.rahman@interdigital.com

Dirk Kutscher
NEC

Email: dirk.kutscher@neclab.eu

Y. Richard Yang
Yale University

Email: yry@cs.yale.edu

Haibin Song
Huawei

Email: haibin.song@huawei.com

Kostas Pentikousis
Huawei

Email: k.pentikousis@huawei.com

DECADE
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

X. Chen
Z. Huang
HUAWEI Technologies
L. Chen
H. Liu
Yale University

Oct 31, 2011

Integration Examples of DECADE System
draft-ietf-decade-integration-example-02

Abstract

DECADE is an in-network storage infrastructure which is under discussions and constructions. It can be integrated into Peer-to-Peer (P2P) applications to achieve more efficient content distributions. This document represents two detailed examples of how to integrate DECADE into P2P applications (live streaming and file sharing). Specifically, it describes mainly about: 1) a preliminary DECADE client API; 2) a P2P live streaming integration with DECADE; 3) a P2P file sharing integration with DECADE; 4)ALTO+DECADE based file distribution platform; 5) tests on our DECADE integrarions and 6) an application performance analysis from the tests.

Table of Contents

1. Introduction 4
2. Concepts 4
2.1. P2P 4
2.2. DECADE Server 4
2.3. DECADE Module 5
2.4. P2P LiveStreaming Client (P2PLS Client) 5
2.5. DECADE Client 5
2.6. Vuze 5
2.7. DECADE Plugin 5
2.8. DECADE-Enabled Vuze 5
2.9. Remote Controller 5
3. DECADE Client API 6
4. DECADE Integration of P2P LiveStreaming Client 6
4.1. DECADE Integration Architecture 7
4.1.1. Data Access 7
4.1.2. Message Control 7
4.2. Challenges in DECADE Integration 8
4.2.1. Limited Connection Slot 8
4.2.2. Additional Control Latency 8
5. DECADE Integration of P2P Filesharing Client 9
5.1. Vuze Client Design 9
5.2. DECADE-Enabled Vuze architecture Design 9
5.3. DECADE-Enabled Vuze Communication Procedure 11
6. ALTO+DECADE based file distribution platform 11
6.1. File distribution platform architecture Design 11
6.2. CP Uploading and Publishing Communication Procedure . . . 11
6.3. User Downloading Communication Procedure 11
7. Test Environment and Settings 12
7.1. Test Settings 13
7.2. Platforms and Components of P2PLS 13
7.2.1. EC2 DECADE Server 14
7.2.2. PlanetLab P2P LiveStreaming Client 14
7.2.3. Tracker 14

7.2.4. Source Server 14
7.2.5. Test Controller 15
7.3. Platforms and Components of Vuze 15
7.3.1. EC2 DECADE Server 16
7.3.2. Vuze Client with DECADE Plugin 16
7.3.3. Remote Controller 16
7.3.4. Tracker 16
7.3.5. HTTP Server 16
7.3.6. PL Manager 16
8. Performance Analysis 17
8.1. Performance Metrics 17
8.1.1. P2P Live Streaming 17
8.1.2. Vuze 17
8.1.3. ALTO+DECADE based file distribution platform 17
8.2. Result and Analysis 17
8.2.1. P2P Live Streaming 17
8.2.2. Vuze 18
8.2.3. ALTO+DECADE based file distribution platform 18
9. Security Considerations 20
10. IANA Considerations 20
11. Normative References 20
Authors' Addresses 20

1. Introduction

DECADE is an in-network storage infrastructure under discussions and constructions. It can be integrated into Peer-to-Peer (P2P) applications to achieve more efficient content distributions.

This draft introduces an instance of application integration with DECADE. In our example system, the core component includes DECADE servers and DECADE-enabled P2P live streaming clients. A DECADE server provides data storage and transport service with interactive control to DECADE clients. We extended a P2P live streaming application, P2PLS (P2P LiveStreaming), to leverage DECADE service provided by the servers. P2PLS clients are DECADE-enabled. For our implementation, we used a preliminary API (Application Programming Interface) set, which is supposed to be provided by DECADE, to enable P2PLS clients to use DECADE in their data transmission. In this draft, we introduce the design of the DECADE-P2PLS integration system, the main control flow for DECADE-enabled data transmission, and the testing performance with this system.

Please note that P2PLS in this draft only represents the usage case of "live streaming" out of a large number of P2P applications, while DECADE itself can support other applications. The API set used in this system is an experimental design and implementation. It is not a standard and is still under development. Currently, DECADE in this draft is only a preliminary framework of in-network storage for P2P. It is designed to test the pros and cons of in-network storage utilized by P2P applications rather than to reach a final solution.

2. Concepts

2.1. P2P

Peer-to-Peer computing or networking is a distributed application architecture that partitions tasks or work loads between peers. Peers are equally privileged, equipotent participants in the application.

2.2. DECADE Server

A DECADE server is implemented with DECADE protocols, management mechanism and storage strategies. It is an important element to provide DECADE services. In a DECADE server, we have a number of Data Lockers each of which is a virtual account and private STORAGE space for applications.

2.3. DECADE Module

DECADE module is a functional component for application clients to utilize DECADE. This component serves as an application-specific interface between a particular application and DECADE servers. It can be a simple implementation of basic DECADE access APIs, or a smart realization which integrates application-specific control strategies with DECADE APIs.

2.4. P2P LiveStreaming Client (P2PLS Client)

P2P LiveStreaming Client (P2PLS Client) is our self-maintained version of a native P2P live streaming application. It is one example out of a large number of P2P applications.

2.5. DECADE Client

DECADE client is an integration of a native P2P client and a DECADE module. It is not required to embed DECADE module into native P2P clients, since it can also be an independent conponent running at a remote server.

2.6. Vuze

Vuze is an open source P2P application, which uses BitTorrent protocol for message and data exchanging. Vuze provides a set of interfaces which support users to develop particular extensions.

2.7. DECADE Plugin

A plugin built into Vuze to implement DECADE functions including getting/putting data from/to DECADE server and redirection

2.8. DECADE-Enabled Vuze

A Vuze client that is enabled by DECADE plugin.

2.9. Remote Controller

A controller which can control every Vuze client to start or stop downloading tasks. It also has a function of collecting statistic information of each Vuze client. It is a major operating platform.

3. DECADE Client API

In order to simplify the DECADE integration with P2P clients, we provide an API set which covers the communications with DECADE servers and token-generation. On top of this API, a P2P client can develop its own application-specific control and data distribution policies.

There are five basic interfaces:

o Get_Object: to get an object from a DECADE server with an authorized token. The get operation can be classified into two categories: Local Get and Remote Get. Local Get is to get an object from a local DECADE server. Remote Get is to use a client's local DECADE server to indirectly get an object from a remote DECADE server. The object will be firstly passed to the local DECADE server, then returned to the application client.

o Put_Object: to store an object into a DECADE server with an authorized token. A client can either store an object into its local DECADE server or into other clients' DECADE servers, if it has an authorized token. Both of the operations are direct, for we don't provide indirectly storing (i.e. remote put).

o Delete_Object: to delete an object in a DECADE server explicitly with an authorized token. Note that an object can also be deleted implicitly by setting an expired time or a specific TTL.

o Status_Query: to query current status of an application itself, including listing stored objects, resource usage, etc.. Such information is private.

o Generate_Token: to generate an authorized token. The token can be used to access an application client's local DECADE server, or passed to other clients to allow them to access the client's local DECADE server.

4. DECADE Integration of P2P LiveStreaming Client

We integrate a DECADE module into a P2P live streaming application--P2PLS, in order that clients of this application can easily leverage

4.1. DECADE Integration Architecture

The architecture of the P2PLS application and DECADE integration is shown in Figure 1:

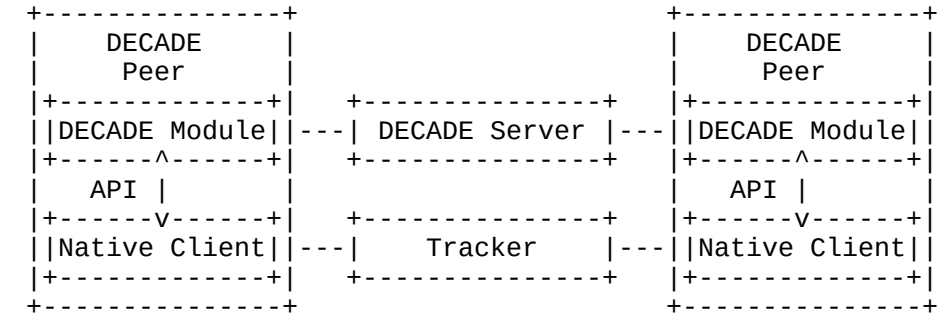


Figure 1

A DECADE-integrated P2PLS client uses DECADE module to communicate with its DECADE server and transmit data between itself and its DECADE server. It is compatible with its original P2P protocol, while it also uses a DECADE protocol to exchange DECADE related messages with other peers.

4.1.1. Data Access

DECADE module is called whenever a client wants to get data objects from (or put data objects into) its DECADE server. Each data object transferred between a client and its DECADE server should go through DECADE module. Neither the DECADE server and the original client knows each other. A data object is a data transfer unit between DECADE servers and application clients. Its size can be application-customized, according to variable requirements of performance or sensitive factors (e.g. low latency, high bandwidth utilization).

4.1.2. Message Control

Control and data plane decoupling is a design principle of DECADE. Control messages are propagated in an original P2P way. DECADE only introduces an additional control message between DECADE module which carries DECADE authorized token. By exchanging DECADE authorized tokens, P2P live streaming clients can retrieve or store data objects into or from others' DECADE servers.

4.2. Challenges in DECADE Integration

One essential objective of DECADE integration is to improve (or at least not to hurt) the application performance. However, as a brand new architecture, DECADE has some inherit challenges which will potentially be harmful to application performance. In our P2P live streaming case, we met mainly two such limitations of DECADE:

4.2.1. Limited Connection Slot

Limited Connection Slot: In native P2P systems, a peer can establish tens or hunderds of concurrent connections with other peers. However, this situation can hardly be ture when it is integrated with DECADE because it is too expensive for DECADE servers to maintain so many connections for each peer. Typically, each DECADE peer only have m connection slots, which means it can only at most have m active connections with its DECADE server simultaneously. A potential "side effect" of limited connection slot is that the content availability and downloading rate might be impacted negatively by fewwer connections carrying data traffic. This requests us to adjust the peer’s behavior in resource allocation, downloading/uploading scheduling, etc. to fully utilize the connection slots to archieve a satisfying and robust data downloading.

o Batch Request: In order to fully utilize the connection bandwidth of a DECADE server and reduce overhead, a P2PLS client may combine multiple requests in a single request to DECADE server. Note that for the sake of improving data transfer efficiency in P2P live streaming, we may combine multiple data requests into a batch request. Generally, a batch may consist of different kinds of access requests.

o Data Object Size: In typical P2P live streaming application, the size of a data block is relatively small, considering to reduce end-to-end transport latency. However, existing data size may incur large control overhead and low transport utilization. A larger data object size may be needed to utilize DECADE more efficiently.

4.2.2. Additional Control Latency

Addtional Control Latency: In native P2P systems, when an uploader decides to reply a request for a piece, it sends the piece out directly. Nevertheless, in DECADE-aware P2P systems, the uploader typically only replies with a token of the piece. And then the downloader will leverage this token to fetch the piece from the uploader’s DECADE server. This process obviously introduces additional control latency compared with native P2P systems. It is

even more serious in latency sensitive applications such as P2P live streaming. We need to consider how to reduce such additional delay or how to compensate the lose with other inherit advantages of DECADE.

o Range Token: One way to reduce request latency is to use range token. A P2PLS client may piggyback a range token when it propagates its bitmap to its neighbors, to indicate that all available pieces in the bitmap are accassable by this range token. Then instead of requesting specific pieces from this client and waiting for response, the neighbors can directly use this range token to access data in DECADE servers. Note that this method not only reduce request latency, but also reduce message overhead.

With these adjustments and strategies, DECADE clients' performance was not impacted by the liminations of DECADE and was even better than native clients' as we will see in following sections.

5. DECADE Integration of P2P Filesharing Client

We integrate DECADE with a popular P2P file-sharing application--Vuze.

5.1. Vuze Client Design

Note that Vuze client is classified into two different kinds - Native Vuze and DECADE-Enabled Vuze. When running Native Vuze, it behaves as ordinary BitTorrent client: some Vuze clients upload data for others to download. When using DECADE-Enabled Vuze, the communication and data exchange processes are changed. The uploader uploads data to its DECADE server, and downloaders download data from DECADE servers. By this means, uplink traffic can be reduced sharply and download performance can be improved. It is beneficial for ISPs to save the last-mile uplink bandwidth.

5.2. DECADE-Enabled Vuze architecture Design

DECADE plugin is a key component of our demo system. It has several interfaces with other components as following:

Interface between DECADE plugin and DECADE server: DECADE plugin can upload and download data from DECADE server. It also includes other functions such as user registration, application registration etc.

Interface between DECADE plugin and Vuze client: DECADE plugin can register a listener to intercept the BitTorrent message such AS "BT_Request" message from or to Vuze client, encapsulate the data from DECADE server into "BT_Piece" message and put the "BT_Piece" message into incoming message queue, read the block/piece data from the disk when seeding (to upload the data to DECADE server), and start or stop the download tasks, all these functions are supported in the Plugin API provided by Vuze.

Interface between DECADE plugins: When DECADE plugin intercepts the "BT_Request" message from other Vuze clients, local DECADE plugin sends "Redirect" message to remote DECADE plugin to authorize it to download the piece data from the DECADE server.

Interface between DECADE plugin and Remote Controller: DECADE plugin registers with Remote Controller after starting up, Remote Controller can control all the Vuze clients to start, stop or resume the download tasks through DECADE plugins.

The system architecture is as follow:

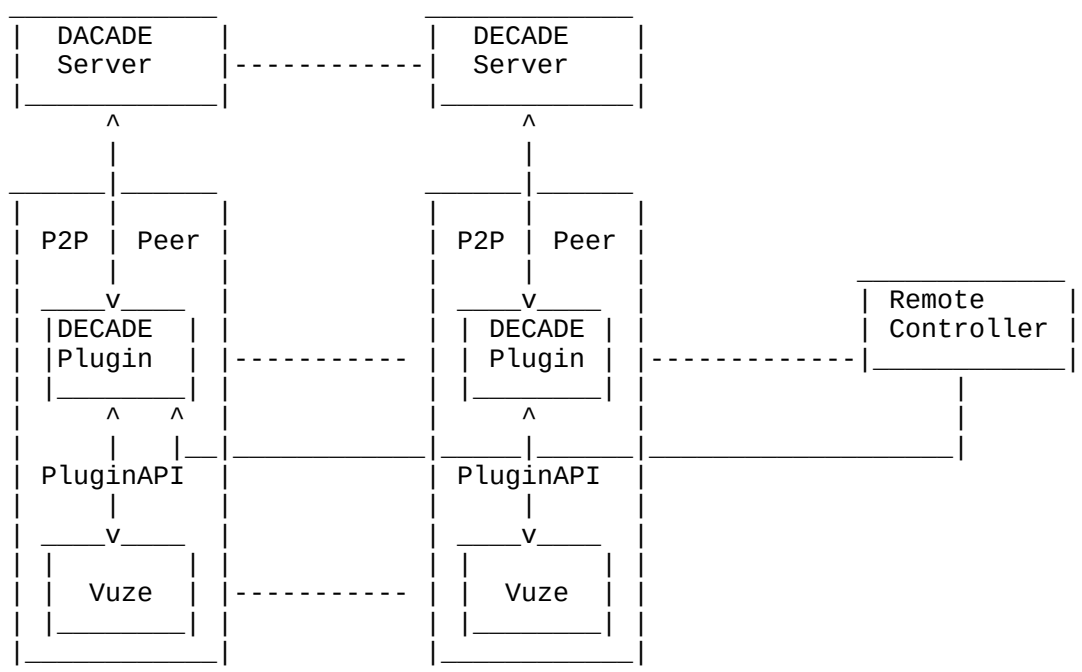


Figure 2

5.3. DECADE-Enabled Vuze Communication Procedure

A DECADE plugin can change the data path of BitTorrent download by using a "Redirect" message.

The detailed communication procedure is as following:

o When each client starts the download task ,it will try to connect the tracker to get peer list and then send "BT_Request" message to other peers in peer list.

o If the DECADE plugin is enabled, then it will intercept the incoming "BT_Request" message from other Vuze clients, and then reply with a "Redirect" message which includes DECADE server's address, authorization token and so on to the requester.

o When a DECADE plugin receives a "Redirect" message, it will connect to the DECADE server according to message context and send "Remote Get" message to the DECADE server to request the data, then wait for response.

o When a DECADE server receives a "Remote Get" message, it will check the server IP address in the message, Case 1: if the address equals to its own IP address, then it will send the data to the requester from its local disk/memory; Case 2: if the address is not equal to its own IP address, then it will send the "Remote Get" message to that address, to fetch the data, and then send the data to the requester. The data will be cached in the server locally for use by other requesters.

o When a DECADE plugin obtains the data, it will encapsulate the data into a standard "BT_Piece" message and send to Vuze client, then the client can write the data block into storage.

The detailed communication diagram is as follow:

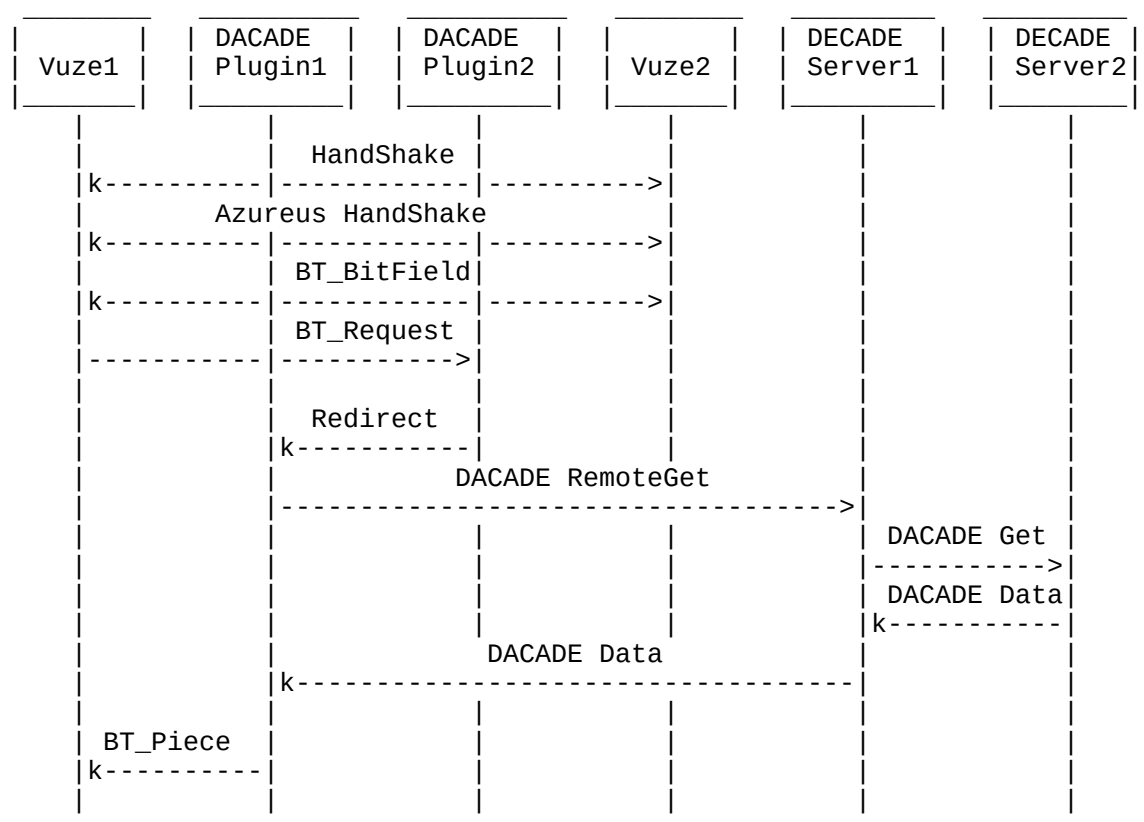


Figure 3

6. ALTO+DECADE based file distribution platform

We integate DECADE with a file distribution platform based on the ALTO+DECADE architecture

6.1. File distribution platform architecture Design

Note that This Integration is a file distribution platform based on the ALTO+DECADE architecture. It allows content providers (CP) to upload files to DECADE servers, and end users to download files from optimal DECADE source servers.Specifically, three components in this trial are provided:

o DECADE Servers:store files.

o DECADEs Portal: offers content publishers a portal site to upload files; it also includes ALTO service to direct end users to an optimal DECADE Server to download files.

o Content Publisher’s Portal: A simulated portal for a content publisher TO offer a web portal to publish download URL and for its END users TO download from DECADE.

The system architecture is as follow:

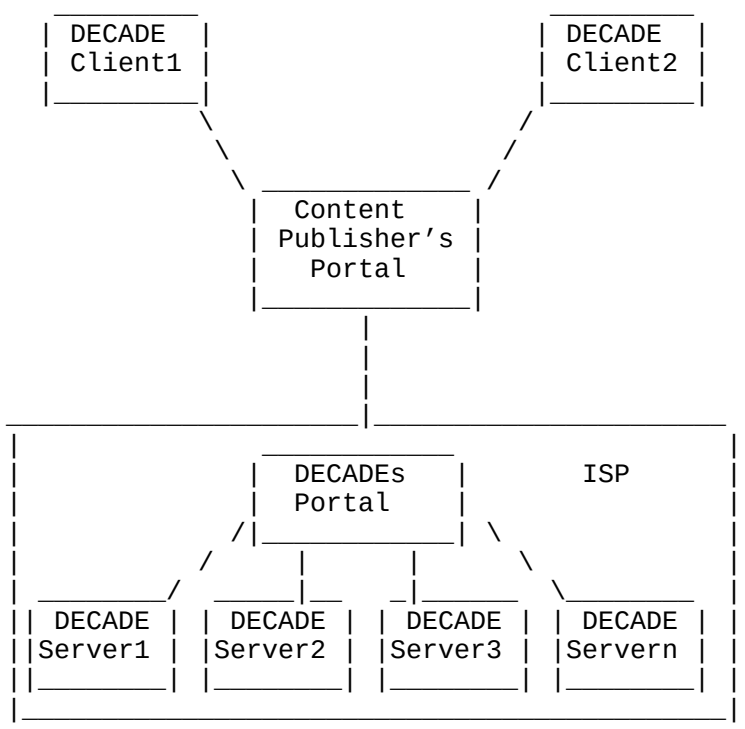


Figure 4

6.2. CP Uploading and Publishing Communication Procedure

Publisher (CP) should upload his file into DECADE Servers first and get the URLs of the files then can publish the download hyperlink on his official homepage.

The detailed communication procedure is as following:

o CP upload any files what he wants to publish via logon DECADE Portal site and send data objects via HTTP flow.

o DECADE Portal can cache the file uploaded by publisher and distribute the file to specified DECADE Sever Zones, DECADE Server will divide the file into many slides as long as the length of file, and save them in different folder.

o When a file has been uploaded successfully, DECADE Portal will list the URL of this file on homepage, publisher can find it easily, and use the URL for publishing.

The detailed communication diagram is as follow:

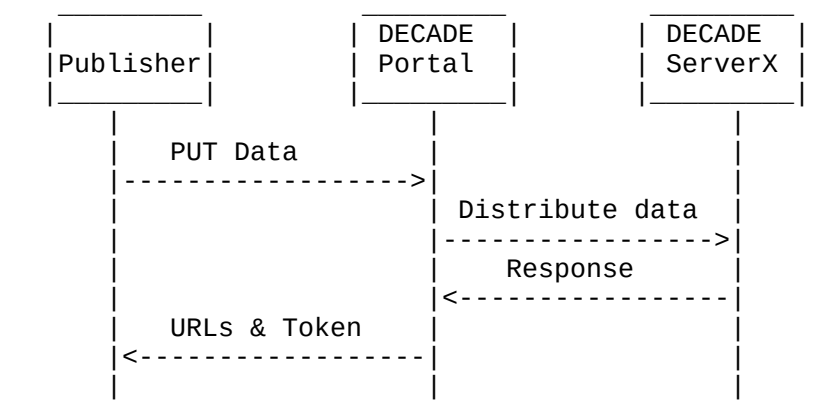


Figure 5

6.3. User Downloading Communication Procedure

When publisher have put file URLs on his publishing homepage, users can visit his downloading page and click hyperlink to download files.Users should visit CP's download list page, and find out which file he is interested in, and then click the hyperlink to download.

The detailed communication procedure is as following:

o User visits publisher's homepage, finds a file download link.

o User clicks the hyper link, homepage returns chunk {URLs & Tokens} to the User, and the user uses those URLs & Tokens to redirect to DECADE Portal's download page.

o DECADE Portal help the user to find an optimal DECADE Server as long as user's IP, and return it to the user.

o user connect to this DECADE Server to get data via HTTP flow.

The detailed communication diagram is as follow:

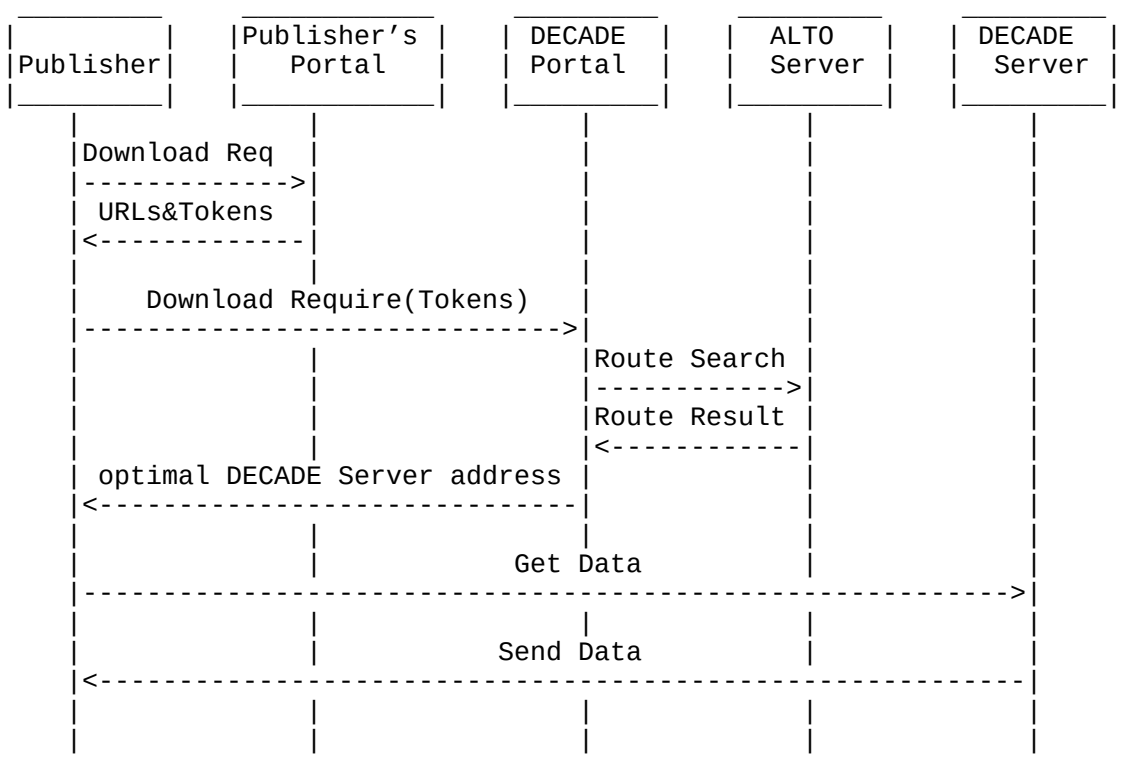

Publisher
Publisher's Portal
DECADE Portal
ALTO Server
DECADE Server
Download Req
URLs&Tokens
Download Require(Tokens)
Route Search
Route Result
optimal DECADE Server address
Get Data
Send Data

Figure 6

7. Test Environment and Settings

In order to demonstrate the performance of our DECADE implementation and DECADE-integrated P2P live streaming and file-sharing applications, we conduct some experimental tests in Amazon EC2 and PlanetLab. We perform a pair of comparative experiments: DECADE integrated P2P application v.s. native P2P application, in the same environment using the same settings.

7.1. Test Settings

Our tests ran on a wide-spread area and diverse platforms, including a famous commercial cloud platform, Amazon EC2 and a well-known testbed, PlanetLab. The environment settings are as following:

o EC2 Regions: we setup DECADE servers in Amazon EC2 cloud, including all four regions around the world, US east, US west, Europe and Asia.

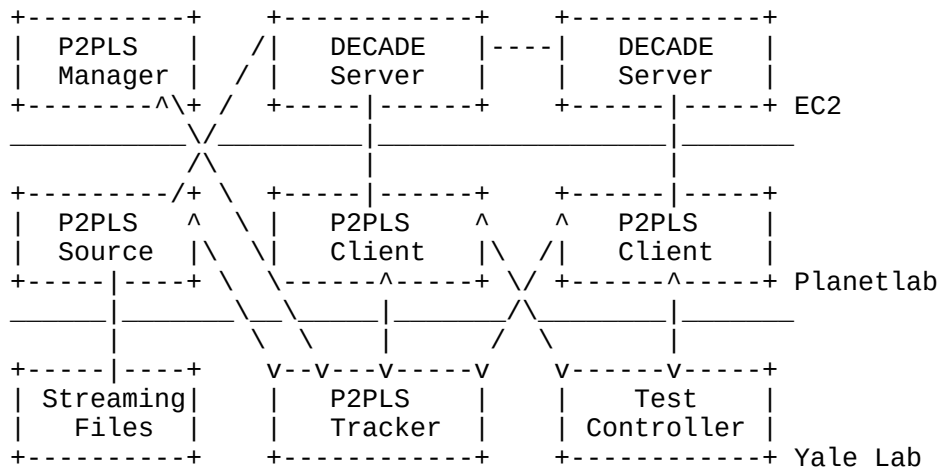
o PlanetLab: we run our P2P live streaming clients and P2P file-sharing clients (both DECADE integrated and native clients) on PlanetLab of a wild-spread area.

o Arrival pattern: we made all the clients join into the system within a short duration to simulate a flash crowd scenario.

o Total Bandwidth: for a fair comparison, we set the system’s total supply bandwidth to be exact the same in both test.

7.2. Platforms and Components of P2PLS

In the tests, we have different functional components running in different platforms, including DECADE servers, P2P live streaming clients(DECADE integrated or Native), Tracker, Source server and Test Controller, as shown in Figure 7.



P2PLS represents to P2P LiveStreaming.

Figure 7

7.2.1. EC2 DECADE Server

DECADE Servers ran on Amazon EC2 small instances, with bandwidth constraint.

7.2.2. PlanetLab P2P LiveStreaming Client

Both DECADE integrated and Native P2P live streaming clients ran on planetlab which spreads in various locations around the world. The DECADE integrated P2P live streaming clients connect to the closest DECADE server according to its Geo-location distance to the servers. DECADE integrated P2P live streaming clients use their DECADE servers to upload to neighbors, instead of their own "last-mile" bandwidth.

7.2.3. Tracker

A native P2P live streaming tracker ran at Yale’s laboratory and served both DECADE integrated clients and native clients during the test.

7.2.4. Source Server

A native P2P live streaming source server ran at Yale’s laboratory and serve both DECADE integrated clients and native clients during the test. The capacity of source is equivalently constrain for both cases.

7.2.5. Test Controller

Test Controller is a manager to control all machines' behaviors in both EC2 and PlanetLab during the test.

7.3. Platforms and Components of Vuze

Test platforms includes Vuze client, tracker, DECADE Plugin, DECADE Servers and other supportive components such as HTTP Server, FTP Server and Remote Controller.

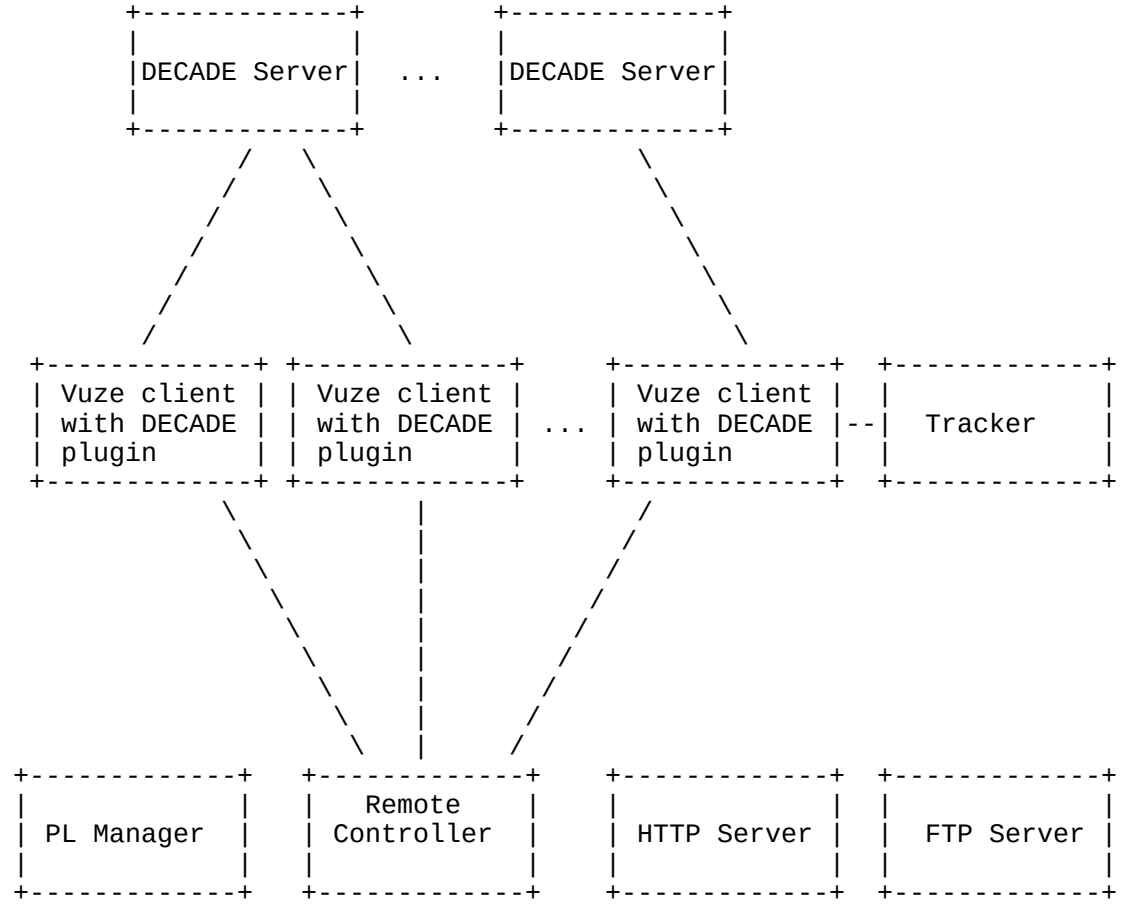


Figure 8

7.3.1. EC2 DECADE Server

DECADE Servers ran on Amazon EC2 small instances, with bandwidth constraint.

7.3.2. Vuze Client with DECADE Plugin

Vuze clients are divided into one seeding client and multiple leechers. Leechers run at PlanetLab, while the seeding client runs at the Window 2003 server. DECADE Plugin will be automatically loaded and run after Vuze client starts up.

7.3.3. Remote Controller

Remote controller can list all the Vuze clients in user interface and control them to download a specific BitTorrent file. It runs at the same Window 2003 server with the seeding client.

7.3.4. Tracker

Vuze client provides tracker capability, so we did not deploy our own tracker. Vuze embedded tracker is enabled when making a torrent file, the seeding client is also a tracker in this test.

7.3.5. HTTP Server

Torrent file will be put in the HTTP Server and the leechers will retrieve the torrent file from HTTP Server after receiving the download command from Remote Controller. We use Apache Tomcat which is an open source software as HTTP Server.

7.3.6. PL Manager

PL Manager (PlanetLab experiment Manager)is a tool developed by University of Washington, which presents a simple GUI to control PlanetLab nodes and perform common tasks such as:

o Selecting nodes for your slice.

o Choosing nodes for your experiment based on CoMon information about the nodes.

o Reliably deploying you experiment files.

o Executing commands / sets of commands on every node in parallel.

o Monitoring the progress of the experiment as a whole, as well as viewing console output from the nodes.

8. Performance Analysis

During the test, Both DECADE integrated P2P live streaming clients and DECADE integrated P2P file-sharing clients achieved impressively better performance than native clients.

8.1. Performance Metrics

8.1.1. P2P Live Streaming

To measure the performance of a P2P live streaming client, we employ mainly four metrics:

o Startup Delay: the duration from a peer joins the channel to the moment it starts to play.

o Piece Missed Rate: the number of pieces a peer loses when playing over the total number of pieces.

o Freeze Times: the number of times a peer re-buffers during playing.

o Average Peer Uploading Rate: Average uploading bandwidth of a peer.

8.1.2. Vuze

For the performance comparison of Native Vuze and DECADE-Enabled Vuze, the same system bandwidth is provided through some parameter settings. In Native Vuze, the system bandwidth is defined as the amount of the uplink bandwidth from all the Vuze clients. The maximum upload bandwidth (Bu) is configured to every Vuze client before the test. Suppose the number of the Vuze clients is N, the system bandwidth is Bu * N. In the DECADE-Enabled Vuze case, the same bandwidth (Bu * N) is configured to the corresponding Ethernet port of DECADE Server.

8.1.3. ALTO+DECADE based file distribution platform

In ALTO+DECADE based file distribution platform, the system bandwidth is defined as the amount of the download bandwidth from all clients. The maximum upload bandwidth is configured to every client before the test. The maximum online users IS defined as the amount of the clients downloading simultaneously.

8.2. Result and Analysis

8.2.1. P2P Live Streaming

o Startup Delay: In the test, DECADE integrated P2P live streaming clients startup around 35˜40 seconds. some of them startup at about 10 seconds. Native P2P live streaming clients startup around 110˜120 seconds. less than 20% of them startup within 100 seconds.

o Piece Missed Rate: In the test, both DECADE integrated P2P live streaming clients and native P2P live streaming clients achieved a good performance in pieces missed rate. Only about 0.02% of total pieces missed in both cases.

o Freeze Times: In the test, native P2P live streaming clients suffered from more freezing than DECADE Integrated P2P live streaming clients by 40%.

o Average Peer Uploading Rate: In the test, according to our settings, DECADE integrated P2P live streaming clients had no upload in their "last-mile" access network. While in the native P2P live streaming system, more than 70% of peers uploaded in a rate that is much more than streaming rate. In another word, DECADE can shift uploading traffic from clients' "last-mile" to in-network devices, which saves a lot of expensive bandwidth on access links..

8.2.2. Vuze

Performance advantage is shown according to the test result of experiment:

o There is no upload data traffic from Vuze clients in the DECADE-Enabled Vuze experiment, but in the Native Vuze experiment, the upload data traffic from Vuze clients is the same as download data traffic. Bandwidth resource is reduced in the last mile in the DECADE-Enabled Vuze experiment. The test result is illustrated in the following table. The download traffic and upload traffic include data traffic and signal traffic. For the upload traffic in the DECADE-enable Vuze, the data traffic is zero so the all traffic is signal overhead. Though we used the same torrent file in those two cases, the number of Vuze client was not the same because the nodes in the Planet-lab are not always stable. Therefore, the download traffic is a little different in two cases.

	Download traffic	Upload traffic
DECADE-Enabled Vuze	480MB	12MB
Native Vuze	430MB	430MB

Figure 9

o Higher system resource efficiency in the DECADE-Enabled Vuze experiment, system resource efficiency is defined as the ratio of system download rate to the total system bandwidth. The test result is illustrated in the following figure.

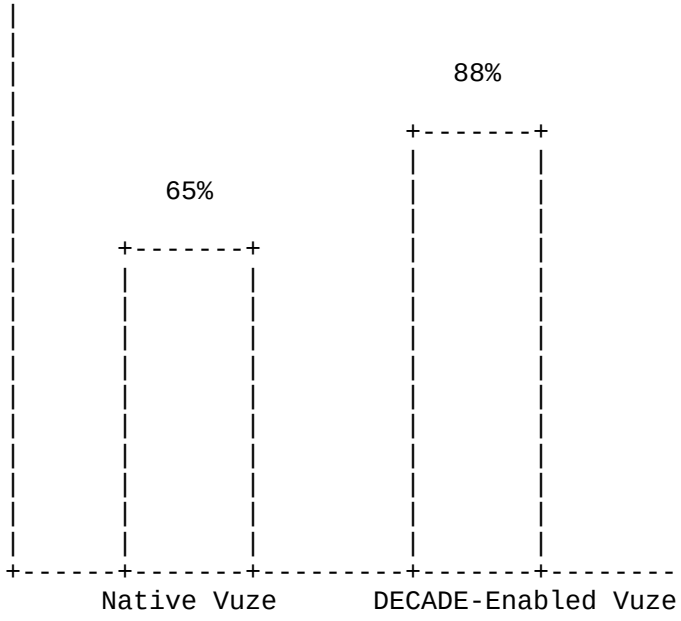


Figure 10

8.2.3. ALTO+DECADE based file distribution platform

- Each DECADE Server can supply transmit bandwidth the same as at most 94% of network interface card (e.g. 1000M interface card server can supply bandwidth of 940Mbps at most).

- Each DECADE Server can support about 400 users online download simultaneously as designed.

9. Security Considerations

This document does not contain any security considerations.

10. IANA Considerations

This document does not have any IANA considerations.

11. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Authors' Addresses

Xiaohui Chen
HUAWEI Technologies

Email: risker.chen@huawei.com

Zhigang Huang
HUAWEI Technologies

Email: hpanda@huawei.com

Lijiang Chen
Yale University

Email: lijiang.chen@yale.edu

Hongqiang Liu
Yale University

Email: hongqiang.liu@yale.edu

DECADE
Internet-Draft
Intended status: Informational
Expires: January 10, 2013

N. Zong, Ed.
X. Chen
Z. Huang
Huawei Technologies
L. Chen
HP Labs
H. Liu
Yale University

July 9, 2012

Integration Examples of DECADE System
draft-ietf-decade-integration-example-06

Abstract

Decoupled Application Data Enroute (DECADE) system is an in-network storage infrastructure which is still under discussion in IETF. This document presents two detailed examples of how to integrate such in-network storage infrastructure into peer-to-peer (P2P) applications to achieve more efficient content distribution, and Application Layer Traffic Optimization (ALTO) system to build a content distribution platform for Content Providers (CPs).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 10, 2013.

Copyright Notice

Table of Contents

1. Introduction 5
2. Terminology 6
2.1. Native Application Client 6
2.2. INS Server 6
2.3. INS Client 6
2.4. INS Operations 6
2.5. INS System 6
2.6. INS Client API 6
2.7. INS-enabled Application Client 6
2.8. INS Service Provider 6
2.9. INS Portal 6
3. INS Client API 7
4. Integration of P2P File Sharing and INS System 7
4.1. Integration Architecture 7
4.1.1. Message Flow 8
4.2. Concluding Remarks 10
5. Integration of P2P Live Streaming and INS System 10
5.1. Integration Architecture 10
5.1.1. Data Access Messages 10
5.1.2. Control Messages 10
5.2. Design Considerations 11
5.2.1. Improve Efficiency for Each Connection 11
5.2.2. Reduce Control Latency 11
6. Integration of ALTO and INS System for File Distribution . . . 12
6.1. Architecture 12
6.1.1. CP Uploading Procedure 13
6.1.2. End User Downloading Procedure 14
7. Test Environment and Settings 15
7.1. Test Settings 15
7.2. Test Environment for P2P Live Streaming Example 15
7.2.1. INS Server 16
7.2.2. P2P Live Streaming Client 16
7.2.3. Tracker 16
7.2.4. Streaming Source Server 16
7.2.5. Test Controller 16
7.3. Test Environment for P2P File Sharing Example 17
7.3.1. INS Server 17
7.3.2. Vuze Client 17
7.3.3. Tracker 17
7.3.4. Test Controller 17
7.3.5. HTTP Server 18
7.3.6. PlanetLab Manager 18
7.4. Test Environment for Combined ALTO and INS File Distribution System 18
8. Performance Analysis 18
8.1. Performance Metrics 18

8.1.1. P2P Live Streaming 18
8.1.2. P2P File Sharing 19
8.1.3. Integration of ALTO and INS System for File Distribution 19
8.2. Results and Analysis 19
8.2.1. P2P Live Streaming 19
8.2.2. P2P File Sharing 20
8.2.3. Integrated ALTO and INS System for File Distribution 21
9. Conclusion 21
10. Security Considerations 21
11. IANA Considerations 21
12. References 21
12.1. Normative References 22
12.2. Informative References 22
Authors' Addresses 22

1. Introduction

Decoupled Application Data Enroute (DECADE) system is an in-network storage infrastructure which is still under discussion in IETF. We implemented such in-network storage infrastructure to simulate DECADE system including DECADE servers, DECADE clients and DECADE protocols [I-D.ietf-decade-arch]. Therefore, in the whole draft, we use the terms of in-network storage (INS) system, INS server, INS client, INS operations, etc.

This draft introduces some examples of integrating INS system with existing applications. In our example systems, the core components include INS server and INS-enabled application client. An INS server stores data inside the network, and thereafter manages both the stored data and access to that data. An INS-enabled application client including INS client and native application client uses a set of Application Programming Interfaces (APIs) to enable native application client to utilize INS operations such as data get, data put, storage status query, etc.

This draft presents two detailed examples of how to integrate INS system into peer-to-peer (P2P) applications, i.e. live streaming and file sharing, as well as an example integration of Application Layer Traffic Optimization (ALTO) [I-D.ietf-alto-protocol] and INS system to support file distribution. We show how to extend native P2P applications by designing the INS-enabled P2P clients and describing the corresponding flows of INS-enabled data transmission. Then we introduce the functional architecture and working flows of integrated ALTO and INS system for file distribution of Content Providers (CPs). Finally we illustrate the performance gain to P2P applications and more efficient content distribution by effectively leveraging the INS system.

Please note that the P2P applications mentioned in this draft only represent some cases out of a large number of P2P applications, while the INS system itself can support a variety of other applications. Moreover, the set of APIs used in our integration examples is an experimental implementation, which is not standard and still under development. The INS system described in this draft is only a preliminary functional set of in-network storage infrastructure for applications. It is designed to test the pros and cons of INS system utilized by P2P applications and verify the feasibility of utilizing INS system to support content distribution. We hope our examples would be useful for further standard protocol design, rather than to present a solution for standardization purpose.

2. Terminology

The following terms will be used in this document.

2.1. Native Application Client

A client running original application operations including control and data messages defined by applications.

2.2. INS Server

A server to simulate DECADE server defined in [I-D.ietf-decade-arch].

2.3. INS Client

A client to simulate DECADE client defined in [I-D.ietf-decade-arch].

2.4. INS Operations

A set of communications between INS server and INS client to simulate DECADE protocols defined in [I-D.ietf-decade-arch].

2.5. INS System

A system including INS servers, INS clients, and INS operations.

2.6. INS Client API

A set of APIs to enable native application client to utilize INS operations.

2.7. INS-enabled Application Client

An INS-enabled application client includes INS client and native application client communicating through INS client API.

2.8. INS Service Provider

An INS service provider deploys INS system and provides INS service to applications/end users. It can be Internet Service Provider (ISP) or other parties.

2.9. INS Portal

A functional entity operated by INS service provider to offer applications/end users a point to access (e.g. upload, download) files stored in INS servers.

3. INS Client API

In order to simplify the integration of INS system with P2P applications, we provide INS client API to native P2P clients for accomplishing INS operations such as data get, data put, etc. On top of the INS client API, a native P2P client can develop its own application specific control and data distribution flows.

We currently developed the following five basic interfaces.

o Generate_Token: Generate an authorization token. An authorization token is usually generated by an entity that is trusted by an INS client which is sharing its data and passed to the other INS clients for data access control. Please see [I-D.ietf-decade-arch] for more details.

o Get_Object: Get a data object from an INS server with an authorization token.

o Put_Object: Store a data object into an INS server with an authorization token.

o Delete_Object: Delete a data object in an INS server explicitly with an authorization token. Note that a data object can also be deleted implicitly by setting a Time-To-Live (TTL) value.

o Status_Query: Query current status of an application itself, including listing stored data objects, resource (e.g. storage space) usage, etc.

4. Integration of P2P File Sharing and INS System

We integrate an INS client into Vuze - a BitTorrent based file sharing application [VzApp].

4.1. Integration Architecture

The architecture of the integration of Vuze and INS system is shown in Figure 1. An INS-enabled Vuze client uses INS client to communicate with INS server and transmit data between itself and INS server. It is also compatible with original Vuze signaling messages such as peer discovery, data availability announcement, etc. Note that the same architecture applies to the other example of integration of P2P live streaming and INS system.

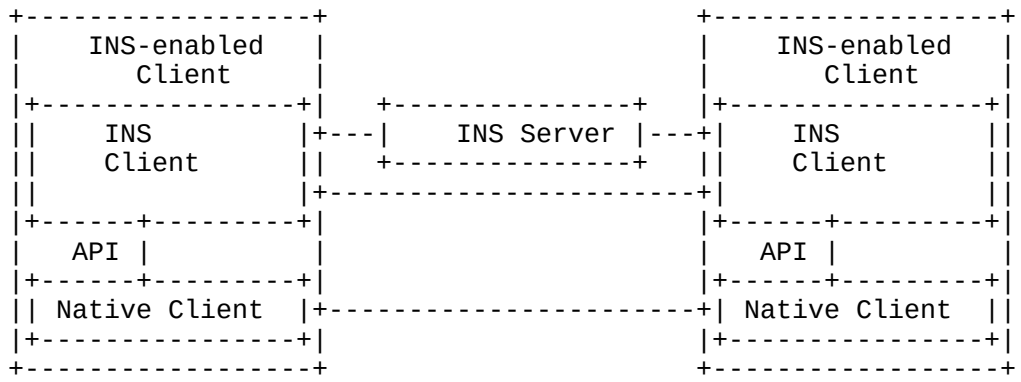


Figure 1

4.1.1. Message Flow

In order for a better comparison, we briefly show the below diagram of the native Vuze message exchange, and then show the corresponding diagram including the INS system.

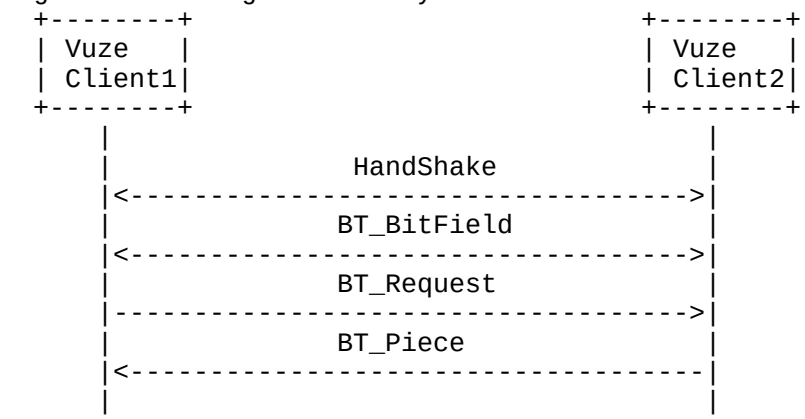


Figure 2

In the above diagram, one can see that the key messages for data sharing in native Vuze are "BT_BitField", "BT_Request" and "BT_Piece". Vuze client1 and client2 exchange "BT_BitField" messages to announce the available data objects to each other. If Vuze client1 wants to get certain data object from client2, it sends a "BT_Request" message to client2. Vuze client2 then return the requested data object to client1 by a "BT_Piece" message. Please refer to [VzMsg] for the detailed description of Vuze messages.

As shown in the below diagram, in the integration of Vuze and INS

system, INS client inserts itself into the Vuze client by intercepting certain Vuze messages, and adjusting their handling to send/receive data using the INS operations instead.

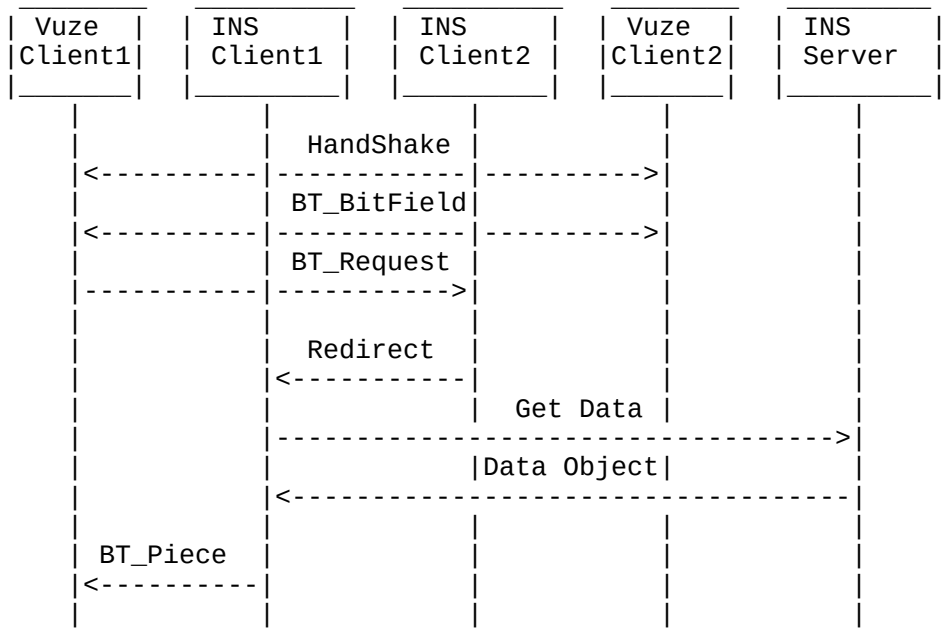


Figure 3

o Vuze client1 sends a "BT_Request" message to Vuze client2 to request a data object as usual.

o INS client2 embedded in Vuze client2 intercepts the incoming "BT_Request" message and then replies with a "Redirect" message which includes INS server's address and authorization token.

o INS client1 receives the "Redirect" message and then sends an INS message "Get Data" to the INS server to request the data object.

o INS server receives the "Get Data" message and sends the requested data object back to INS client1 after the token check.

o INS client1 encapsulates the received data object into a "BT_Piece" message and sends to Vuze client1.

In this example, the file to be shared is divided into many objects, with each object being named as "filename_author_partn" where author is the original author of the file or the user who uploads the file, n is the sequence number of the object.

4.2. Concluding Remarks

In this example, we feel that the INS system can effectively improve the file sharing efficiency due to following reasons: 1) utilizing in-network storage as the data location of the peer will achieve statistical multiplexing gain of the data sharing; 2) shorter data delivery path based on in-network storage could not only improve the application performance, but avoid the potential bottleneck in the ISP network.

5. Integration of P2P Live Streaming and INS System

We integrate an INS client into a P2P live streaming application.

5.1. Integration Architecture

The architecture of the integration of P2P live streaming application and INS system is shown in Figure 1. An INS-enabled P2P live streaming client uses INS client to communicate with INS server and transmit data between itself and INS server.

5.1.1. Data Access Messages

INS client API is called whenever an INS-enabled P2P live streaming client wants to get data objects from (or put data objects into) the INS server. Each data object transferred between the application client and the INS server should go through the INS client. Each data object can be a variable-sized block to cater to different application requirements (e.g. latency and throughput).

We use the hash of a data object’s content for the name of the data object. The name of a data object is generated and distributed by the source streaming server in this example.

5.1.2. Control Messages

We used a lab-based P2P live streaming system for research purpose only. The basic control messages between the native P2P live streaming clients are similar to Vuze control protocols in the sense that the data piece information is exchanged between the peers. The INS-enabled P2P live streaming client adds an additional control message for authorization token distribution, as shown as the line between the INS clients in Figure 1. In this example, the authorization token is generated by the INS client that is sharing its data. By exchanging the authorization tokens, the application clients can retrieve the data objects from the INS servers.

5.2. Design Considerations

One essential objective of the integration is to improve the performance of P2P live streaming application. In order to achieve such goal, we have some important design considerations that would be helpful to the future work of protocol development.

5.2.1. Improve Efficiency for Each Connection

In a native P2P system, a peer can establish tens or hundreds of concurrent connections with other peers. On the other hand, it may be expensive for an INS server to maintain many connections for a large number of INS clients. Typically, each INS server may only allocate and maintain M connections (in our examples, M=1) with each INS client at a time. Therefore, we have the following design considerations to improve the efficiency for each connection between INS server and INS client to achieve satisfying data downloading performance.

o Batch Request: In order to fully utilize the connection bandwidth of INS server and reduce the overhead, an application client may request a batch of data objects in a single request.

o Larger Data Object: Data object size in existing P2P live streaming application may be small and thus incur large control overhead and low transport utilization. A larger data object may be needed to more efficiently utilize the data connection between INS server and INS client.

5.2.2. Reduce Control Latency

In a native P2P system, a serving peer sends data objects to the requesting peer directly. Nevertheless, in an INS system, the serving client typically only replies with an authorization token to the requesting client, and then the requesting client uses this token to fetch the data objects from the INS server. This process introduces an additional control latency compared with the native P2P system. It is even more serious in latency sensitive applications such as P2P live streaming. Therefore, we need to consider how to reduce such control latency.

o Range Token: One way to reduce control latency is to use range token. An INS-enabled P2P live streaming client may piggyback a range token when announcing data availability to other peers, indicating that all available data objects are accessible by this range token. Then instead of requesting some specific data object and waiting for the response, a peer can use this range token to access all available data objects that it was permitted to access in

the INS server.

6. Integration of ALTO and INS System for File Distribution

The objective of ALTO service is to give guidance to applications about which content servers to select to improve content distribution performance in an ISP-friendly way (e.g. reducing network usage within the ISP). The core component of ALTO service is called ALTO server which generates the guidance based on the ISP network information. The ALTO protocol conveys such guidance from the ALTO server to the applications. The detailed description of ALTO protocol can be found in [I-D.ietf-alto-protocol].

In this example, we integrate ALTO and INS system to build a content distribution platform for CPs.

6.1. Architecture

The integrated system allows CPs to upload files to INS servers, and guides end users to download files from the INS servers suggested by ALTO service. The architecture diagram is shown as below. Note that this diagram just shows a basic set of connections between the components. Some redirection including that the INS portal redirects end users to the INS servers can also happen between the components.

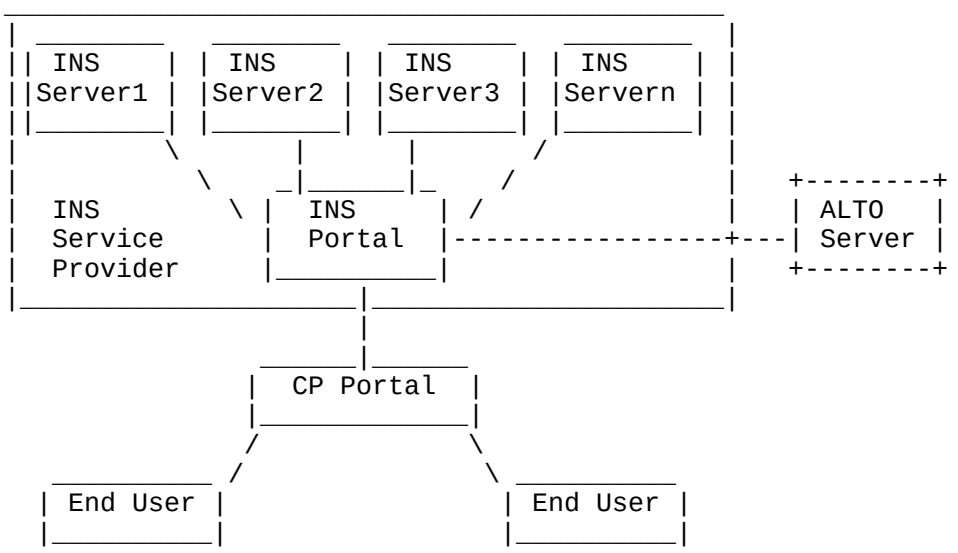


Figure 4

Four key components are defined as follow.

o INS Servers: operated by an INS service provider to store files from CPs.

o INS Portal: operated by an INS service provider to 1) upload files from CPs to the dedicated INS servers; 2) direct end users to the INS servers suggested by ALTO service to download files.

o CP Portal: operated by a CP to publish the URLs of the uploaded files for end user downloading.

o End User: End users use standard web browser with INS extensions such that INS client APIs can be called for fetching the data from INS servers.

6.1.1. CP Uploading Procedure

CP uploads the files into INS servers first, then gets the URLs of the uploaded files and publishes the URLs on the CP portal for end user downloading. The flow is shown below.

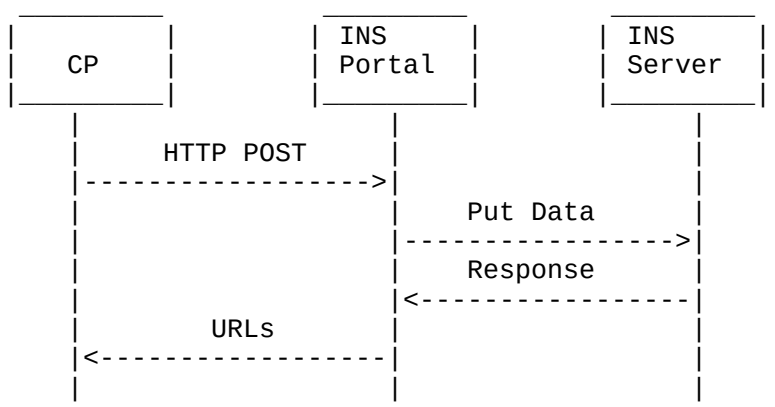


Figure 5

o CP uploads the file to the INS portal site via HTTP POST message.

o INS portal distributes the file to the dedicated INS severs using INS message "Put Data". Note that the data distribution policies (e.g. how many copies of the data to which INS servers) can be specified by CP. The dedicated INS servers can be also decided by the INS service provider based on policies or system status (e.g. INS server load). These issues are out of the scope of this draft.

In this example, the data stored in INS server is divided into many objects, with each object being named as "filename_CPname_partn" where CPname is the name of the CP who uploads the file, n is the

sequence number of the object.

o When the file is uploaded successfully, CP portal will list the URLs of the file for end use downloading.

6.1.2. End User Downloading Procedure

End users can visit the CP portal web pages and click the URLs for downloading the desired files. The flow is shown below.

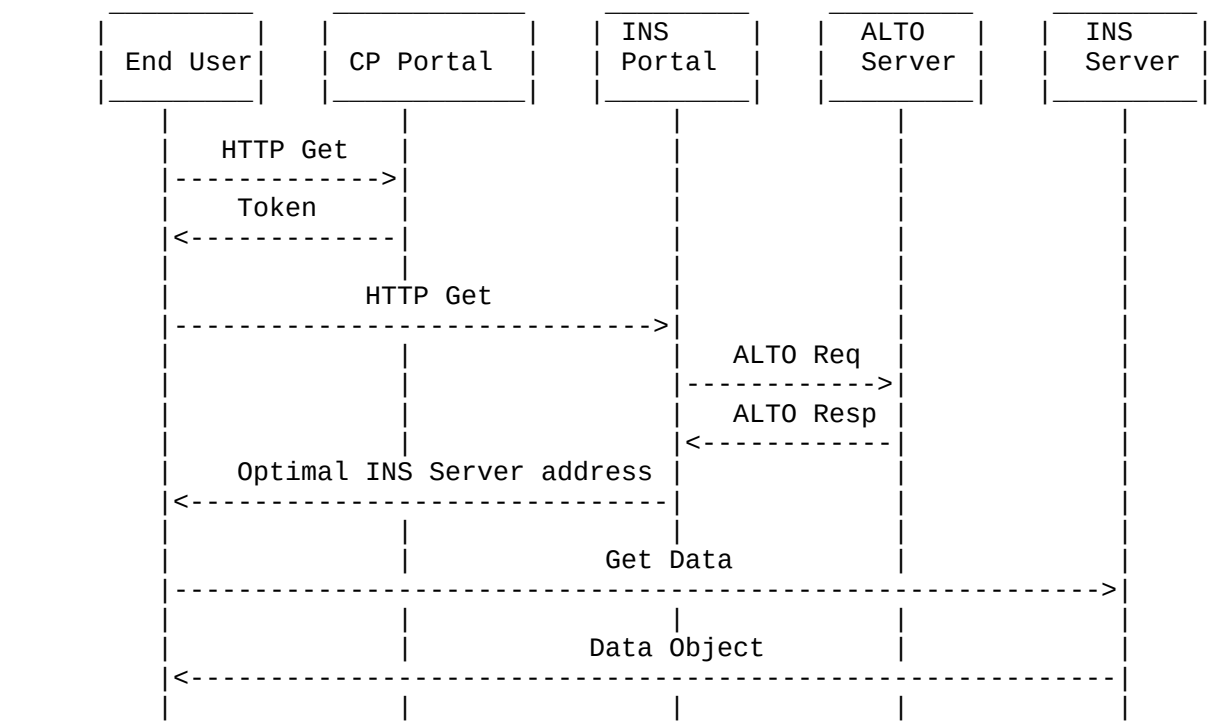


Figure 6

o End user visits CP portal web page, and finds the URLs for the desired file.

o End user clicks the hyper link, CP portal returns authorization token to the end user and redirects the end user to INS portal via HTTP Get message.

o INS portal communicates with ALTO server to get the suggested INS server storing the requested file. In this example, ALTO server just selects the INS server within the same IP subset of the end user. Please see [I-D.ietf-alto-protocol] for more details on how ALTO

select content server.

o INS portal returns the INS server address suggested by ALTO service to the end user.

o End user connects to the suggested INS server to get data via INS message "Get Data" after the token check.

7. Test Environment and Settings

We conduct some tests to show the results of our integration examples. For a better performance comparison, we ran experiments (i.e. INS integrated P2P application v.s. native P2P application) in the same environment using the same settings.

7.1. Test Settings

Our tests ran on a wide-spread area and diverse platforms, including a famous commercial platform - Amazon EC2 [EC2] and a well known test-bed - PlanetLab [PL]. The experimental settings are as follows.

o Amazon EC2: We setup INS servers in Amazon EC2 platform, including four regions around the world - US east, US west, Europe and Asia.

o PlanetLab: We ran our P2P live streaming clients and P2P file sharing clients (both INS-enabled and native clients) on PlanetLab on a wild-spread area.

o Flash-crowd: Flash-crowd is an important scenario in P2P live streaming system due to the live nature, i.e. a large number of users join the live channel during the startup period of the event. Therefore, we conduct experiments to test the system performance for flash-crowd in our P2P live streaming example.

o Total supply bandwidth: Total supply bandwidth is the sum of the capacity of bandwidth used to serve the streaming/file content, from both servers (including source servers and INS servers) and the P2P clients. For a fair comparison, we set the total supply bandwidth to be the same in both tests of native and INS-enabled P2P applications.

7.2. Test Environment for P2P Live Streaming Example

In the tests, we have some functional components running in different platforms, including INS servers, P2P live streaming clients (INS-enabled or native), native P2P live streaming tracker, streaming source server and test controller, as shown in below figure.

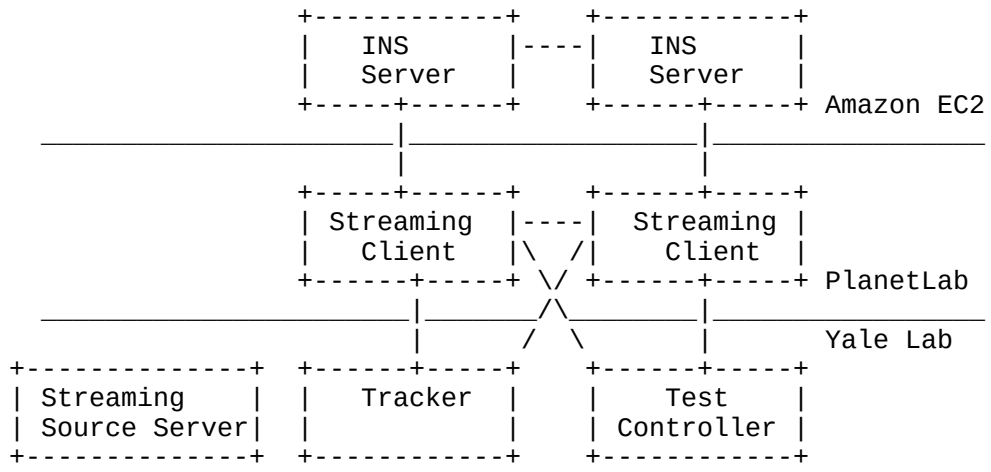


Figure 7

7.2.1. INS Server

INS servers ran on Amazon EC2.

7.2.2. P2P Live Streaming Client

Both INS-enabled and native P2P live streaming clients ran on PlanetLab. Each INS-enabled P2P live streaming client connects to the dedicated INS server. In this example, we decide which client connects to which server based on the IP address. So, it is roughly region-based and still coarse. Each INS-enabled P2P live streaming client uses its INS server to share streaming content to other peers.

7.2.3. Tracker

A native P2P live streaming tracker ran at Yale’s laboratory and served both INS-enabled and native P2P live streaming clients during the test.

7.2.4. Streaming Source Server

A streaming source server ran at Yale’s laboratory and served both INS-enabled and native P2P live streaming clients during the test.

7.2.5. Test Controller

Test controller is a manager running at Yale’s laboratory to control all machines’ behaviors in both Amazon EC2 and PlanetLab during the test.

7.3. Test Environment for P2P File Sharing Example

Functional components include Vuze client (with and without INS client), INS servers, native Vuze tracker, HTTP server, PlanetLab manager and test controller, as shown in below figure.

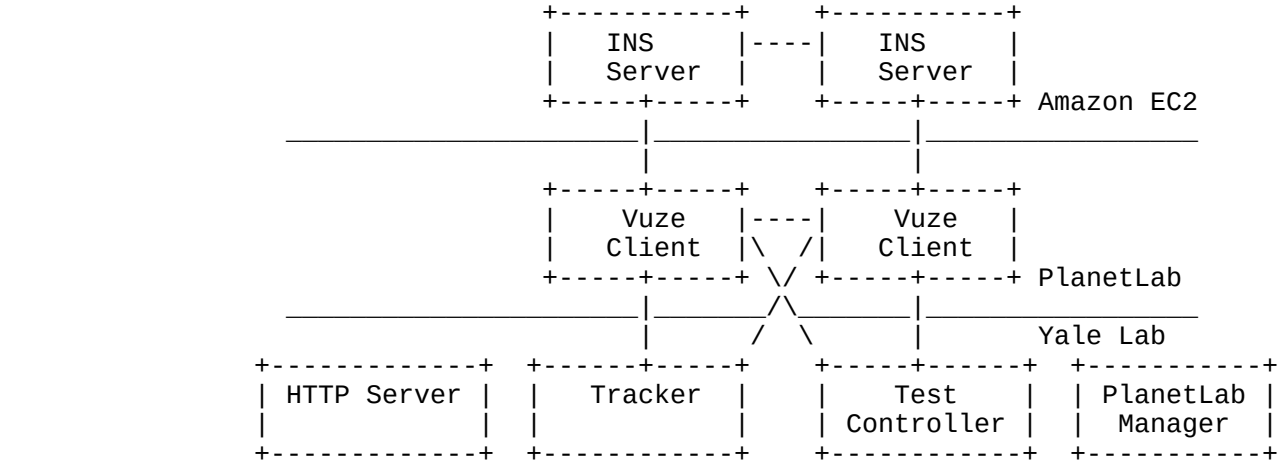


Figure 8

7.3.1. INS Server

INS servers ran on Amazon EC2.

7.3.2. Vuze Client

Vuze clients were divided into one seeding client and multiple leechers. The seeding client ran at a Window 2003 server at Yale’s laboratory. Both INS-enabled and native Vuze clients (leechers) ran on PlanetLab. INS client embedded in Vuze client was automatically loaded and ran after Vuze client start up.

7.3.3. Tracker

Vuze software includes tracker implementation, so we didn’t deploy our own tracker. Tracker ran at Yale’s laboratory and was enabled when making a BitTorrent file. Tracker ran at the same Window 2003 server with the seeding client.

7.3.4. Test Controller

Similar to the test controller in P2P live streaming case, the test controller in Vuze example can also control all machines’ behaviors in Amazon EC2 and PlanetLab. For example, it lists all the Vuze

clients via GUI and controls them to download a specific BitTorrent file. Test controller ran at the same Window 2003 server with the seeding client.

7.3.5. HTTP Server

BitTorrent file was put in the HTTP server and the leechers retrieved the BitTorrent file from the HTTP server after receiving the downloading command from the test controller. We used Apache Tomcat for HTTP server.

7.3.6. PlanetLab Manager

PlanetLab manager is a tool developed by University of Washington. It presents a simple GUI to control PlanetLab nodes and perform common tasks such as: 1) selecting nodes for your slice; 2) choosing nodes for your experiment based on the information about the nodes; 3) reliably deploying you experiment files; 4) executing commands on every node in parallel; 5) monitoring the progress of the experiment as a whole, as well as viewing console output from the nodes.

7.4. Test Environment for Combined ALTO and INS File Distribution System

For the integration of ALTO and INS systems for supporting file distribution of CPs, we built 6 Linux virtual machines (VMs) with Fedora13 operating system. ALTO server, INS portal, CP portal and two INS servers ran on these VMs. Each VM is allocated with 4 cores from a 16-core 1Ghz CPU, and has 2GB memory space and 10GB disk space. CP uploaded files to the INS server via INS portal. End user can choose desired file through the CP portal, and download it from the optimal INS server chosen by the INS portal using ALTO service.

8. Performance Analysis

We illustrate the performance gain to P2P applications and more efficient content distribution by effectively leveraging the INS system. For the example of integrating ALTO and INS systems to support file distribution of CPs, we show the feasibility of such integration.

8.1. Performance Metrics

8.1.1. P2P Live Streaming

To measure the performance of a P2P live streaming application, we mainly employed the following four metrics.

o Startup delay: The duration from a peer joins the streaming channel to the moment it starts to play.

o Piece missed rate: The number of pieces a peer loses when playing over the total number of pieces.

o Freeze times: The number of times a peer re-buffers during playing.

o Average peer uploading rate: Average uploading bandwidth of a peer.

8.1.2. P2P File Sharing

To measure the performance of a P2P file sharing application, we mainly employed the following three metrics.

o Download traffic: The total amount of traffic representing the network downlink resource usage.

o Upload traffic: The total amount of traffic representing the network uplink resource usage.

o Network resource efficiency: The ratio of P2P system download rate to the total network (downlink) bandwidth.

8.1.3. Integration of ALTO and INS System for File Distribution

We consider some common capacity metrics for content distribution system, i.e. the bandwidth usage of each INS server, and the total online users supported by each INS server.

8.2. Results and Analysis

8.2.1. P2P Live Streaming

o Startup delay: In the test, INS-enabled P2P live streaming clients startup around 35˜40 seconds and some of them startup around 10 seconds. Native P2P live streaming clients startup around 110˜120 seconds and less than 20% of them startup within 100 seconds.

o Piece missed rate: In the test, both INS-enabled P2P live streaming clients and native P2P live streaming clients achieved a good performance in piece missed rate. Only about 0.02% of total pieces missed in both cases.

o Freeze times: In the test, native P2P live streaming clients suffered from more freezing times than INS-enabled P2P live streaming clients by 40%.

o Average peer uploading rate: In the test, according to our settings, INS-enabled P2P live streaming clients had no data upload in their "last mile" access network, while in the native P2P live streaming system, most peers uploaded streaming data for serving other peers. In another word, INS system can shift uploading traffic from clients' "last mile" to in-network devices, which saves a lot of expensive bandwidth on access links.

8.2.2. P2P File Sharing

The test result is illustrated in below figure. We can see that there is very few upload traffic from the INS-enabled Vuze clients, while in the native Vuze case, the upload traffic from Vuze clients is the same as the download traffic. Network resource usage is thus reduced in the "last mile" in the INS-enabled Vuze case. This result also verifies that the INS system can shift uploading traffic from clients' "last mile" to in-network devices. Note that because not all clients finish downloading process, there are different total download traffic for the independent tests, as shown in below figure.

	Download Traffic	Upload Traffic
INS-Enabled Vuze	480MB	12MB
Native Vuze	430MB	430MB

Figure 9

We also found higher network resource efficiency in the INS-enabled Vuze case where the network resource efficiency is defined as the ratio of P2P system download rate to the total network (downlink) bandwidth. The test result is that the network resource efficiency of native Vuze is 65% while that of INS-enabled Vuze is 88%. A possible reason behind the higher network resource efficiency is that the INS server can always serve content to the peers, while in traditional P2P applications, peer has to finish downloading content before sharing with other peers.

8.2.3. Integrated ALTO and INS System for File Distribution

Each INS server can supply the bandwidth usage of at most 94% of network interface card (NIC) - e.g. 1Gbps NIC server can supply bandwidth of 940Mbps at most. We did tests on 100Mbps and 1Gbps NIC, and got same result of 94% bandwidth usage.

Each INS server can support about 400 online users for file downloading simultaneously. When we tried 450 concurrent online users, 50 users didn't start downloading on time, but wait for the other 400 users to finish downloading.

9. Conclusion

This document presents two examples of integrating INS system into P2P applications (i.e. P2P live streaming and Vuze) by developing INS client API for native P2P clients. To better adopt INS system, we found some important design considerations including efficiency for INS connection, control latency caused by INS operations, and developed some mechanisms to address them. We ran some tests to show the results of our integration examples on Amazon EC2 and PlanetLab for deploying INS servers and clients, respectively. It can be observed from our test results that integrating INS system into native P2P applications could achieve performance gain to P2P applications and more network efficient content distribution. For the example of integrating ALTO and INS system to support file distribution of CPs, we have shown the feasibility of such integration.

10. Security Considerations

The authorization token can be passed from one INS client to other INS clients to authorize other INS clients to access data objects from its INS storage. Detailed mechanisms of token based authentication and authorization can be found in [I-D.ietf-decade-arch].

11. IANA Considerations

This document does not have any IANA considerations.

12. References

12.1. Normative References

[I-D.ietf-decade-arch] Alimi, R., Yang, Y., Rahman, A., Kutscher, D., and H. Liu, "DECADE Architecture", draft-ietf-decade-arch-07 (work in progress), July 2012.

[I-D.ietf-alto-protocol] Alimi, R., Penno, R., and Y. Yang, "ALTO Protocol", draft-ietf-alto-protocol-11 (work in progress), March 2012.

12.2. Informative References

[VzApp] "http://www.vuze.com"

[VzMsg] "http://wiki.vuze.com/w/Azureus_messaging_protocol"

[EC2] "http://aws.amazon.com/ec2/"

[PL] "http://www.planet-lab.org/"

Authors' Addresses

Ning Zong (editor)
Huawei Technologies

Email: zongning@huawei.com

Xiaohui Chen
Huawei Technologies

Email: risker.chen@huawei.com

Zhigang Huang
Huawei Technologies

Email: andy.huangzhigang@huawei.com

Lijiang Chen
HP Labs

Email: lijiang.chen@hp.com

Hongqiang Liu
Yale University

Email: hongqiang.liu@yale.edu

DECADE
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

Y. Gu
Huawei
D. Bryan
Polycom, Inc.
Y. Yang
Yale University
R. Alimi
Google

October 31, 2011

DECADE Requirements
draft-ietf-decade-reqs-05

Abstract

The target of DECoupled Application Data Enroute (DECADE) is to provide an open and standard in-network storage system for applications, primarily P2P (peer-to-peer) applications, to store, retrieve and manage their data. This draft enumerates and explains requirements, not only for storage and retrieval, but also for data management, access control and resource control, that should be considered during the design and implementation of a DECADE system. These are requirements on the entire system; some of the requirements may eventually be implemented by an existing protocol with/without some extensions (e.g., a protocol used to read and write data from the storage system). The requirements in this document are intended to ensure that the DECADE architecture includes all of the desired functionality for intended applications.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Status of this Memo

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on May 3, 2012.

Copyright Notice

Table of Contents

1. Introduction 5
2. Terminology and Concepts 6
3. Requirements Structure 6
4. Protocol Requirements 7
4.1. Overall Protocol Requirements 7
4.1.1. Connectivity Concerns 7
4.1.1.1. NATs and Firewalls 7
4.1.1.2. Connections to Clients 7
4.1.2. Security 8
4.1.2.1. Secure Transport 8
4.1.3. Error and Failure Conditions 8
4.1.3.1. Overload Condition 8
4.1.3.2. Insufficient Resources 8
4.1.3.3. Unavailable and Deleted Data 9
4.1.3.4. Insufficient Permissions 9
4.1.3.5. Redirection 9
4.2. Transfer and Latency Requirements 9
4.2.1. Low-Latency Access 9
4.2.2. Data Object Size 10
4.2.3. Communication among DECADE Servers 10
4.3. Data Access Requirements 11
4.3.1. Reading/Writing Own Storage 11
4.3.2. Access by Other Users 11
4.3.3. Negotiable Data Transport Protocol 11
4.3.4. Separation of Data and Control Policies 12
4.4. Data Management Requirements 12
4.4.1. Agnostic of reliability 12
4.4.2. Data Object Attributes 12
4.4.3. Time-to-live for Written Data Objects 13
4.4.4. Offline Usage 13
4.5. Data Naming Requirements 13
4.5.1. Unique Names 13
4.6. Resource Control 14
4.6.1. Multiple Applications 14
4.6.2. Per-Remote-Client, Per-Data Control 14
4.6.3. Server Involvement 15
4.7. Authorization 15
4.7.1. Per-Remote-Client, Per-Data Read Access 15
4.7.2. Per-User Write Access 15
4.7.3. Default Access Permissions 16
4.7.4. Authorization Checks 16
4.7.5. Cryptographic Credentials 16
4.7.6. Server Involvement 16
4.7.7. Protocol Reuse 17
4.8. Non-Requirements 17
4.8.1. Application-defined Properties and Metadata 17

5. Storage Requirements 17
5.1. Immutable Data 17
5.2. Explicit Deletion of Data 18
5.3. Multiple writing 18
5.4. Multiple reading 18
5.5. Reading before completely written 18
5.6. Hints concerning usage of written data 19
5.7. Writing model 19
5.8. Storage Status 19
6. Discovery Requirements 20
6.1. Requirements 20
6.1.1. Locating DECADE Servers 20
6.1.2. Support for Clients Behind NATs and Firewalls 20
6.1.3. Prefer Existing Protocols 20
7. Future Considerations 21
7.1. Fairness 21
7.2. Removal of Duplicate Data Objects 21
7.3. Gaming of the Resource Control Mechanism 21
8. Security Considerations 21
8.1. Authentication and Authorization 22
8.2. Encrypted Data 22
9. IANA Considerations 22
10. References 22
10.1. Normative References 22
10.2. Informative References 22
Appendix A. Acknowledgments 23
Authors' Addresses 23

1. Introduction

The object of DECoupled Application Data Enroute (DECADE) is to provide an open and standard in-network storage system for content distribution applications, where data is typically broken into one or more chunks and then distributed. This may already include many types of applications including P2P applications, IPTV (Internet Protocol Television), and VoD (Video on Demand). (For a precise definition of the applications targeted in DECADE, see the definition for Target Application in Section 2.) Instead of always transferring data directly from a source/owner client to a requesting client, the source/owner client can write to and manage its content on its in-network storage. The requesting client can get the address of the in-network storage pertaining to the source/owner client and read data from the storage.

This draft enumerates and explains the rationale behind SPECIFIC requirements on the protocol design and on any data store implementation that may be used to implement DECADE servers that should be considered during the design and implementation of a DECADE system. As such, it DOES NOT include general guiding principles. General design considerations, explanation of the problem being addressed, and enumeration of the types of applications to which DECADE may be suited is not considered in this document. For general information, please see the problem statement [I-D.ietf-decade-problem-statement] and architecture [I-D.ietf-decade-arch] drafts.

This document enumerates the requirements to enable target applications to utilize in-network storage. In this context, using storage resources includes not only basic capabilities such as writing, reading, and managing data, but also controlling access for particular remote clients with which it is sharing data. Additionally, we also consider controlling the resources used by remote clients when they access data as an integral part of utilizing the network storage.

This document discusses requirements pertaining to DECADE protocol(s). In certain deployments, several logical in-network storage systems could be deployed (e.g., within the same administrative domain). These in-network storage systems can communicate and transfer data through internal or non-standard communication messages that are outside of the scope of these requirements, but they SHOULD use DECADE protocol(s) when communicating with other DECADE-capable in-network storage systems.

2. Terminology and Concepts

This document uses terms defined in [I-D.ietf-decade-problem-statement].

This document also defines additional terminology:

Target Application: An application (typically installed at end-hosts) with the ability to explicitly control usage of network and/or storage resources to deliver content to a large number of users. This includes scenarios where multiple applications or entities cooperate, such as with P2P, CDN, and hybrid P2P/CDN architectures. Such applications distribute large amounts of content (e.g., a large file, or video stream) by dividing the content into smaller blocks for more flexible distribution (e.g., over multiple application-level paths). The distributed content is typically immutable (though it may be deleted). We use the term Target Application to refer to the type of applications that are explicitly (but not exclusively) supported by DECADE.

3. Requirements Structure

The DECADE protocol is intended to sit between Target Applications and a back-end storage system. DECADE does not intend to develop yet another storage system, but rather to create a protocol that enables Target Applications to make use of storage within the network, leaving specific storage system considerations to the implementation of the DECADE servers as much as possible. For this reason, we have divided the requirements into two primary categories:

- Protocol Requirements: Protocol requirements for Target Applications to make use of in-network storage within their own data dissemination schemes. Development of these requirements is guided by a study of data access, search and management capabilities used by Target Applications. These requirements may be met by a combination of existing protocols and new protocols.

- Storage Requirements: Functional requirements necessary for the back-end storage system employed by the DECADE server. Development of these requirements is guided by a study of the data access patterns used by Target Applications. These requirements should be met by the underlying data transport used by DECADE. In this document, we use "data transport" to refer to a protocol used to read and write data from DECADE in-network storage.

Note that a third category also enumerates requirements on the protocol used to discover DECADE Servers.

It should also be made clear that the approach is to make DECADE a simple protocol, while still enabling its usage within many Target Applications. For this reason, and to further reinforce the distinction between DECADE and a storage system, in some cases we also highlight the non-requirements of the protocol. These non-requirements are intended to capture behaviors that will NOT be assumed to be needed by DECADE's Target Applications and hence not present in the DECADE protocol.

Finally, some implementation considerations are provided, which while not strictly requirements, are intended to provide guidance and highlight potential points that need to be considered by the protocol developers, and later by implementors.

4. Protocol Requirements

This section details the requirements of DECADE protocol(s).

4.1. Overall Protocol Requirements

4.1.1. Connectivity Concerns

4.1.1.1. NATs and Firewalls

REQUIREMENT(S): DECADE client to server protocols SHOULD be usable across firewalls and NAT (Network Address Translation) devices without requiring additional network support (e.g., Application-level Gateways).

RATIONALE: Firewalls and NATs are widely used in the Internet today, both in ISP (Internet Service Provider) and Enterprise networks and by consumers. Deployment of DECADE must not require modifications to such devices (beyond, perhaps, reconfiguration). Note that this requirement applies to both any new protocol developed by the DECADE Working Group and any data transport used with DECADE.

4.1.1.2. Connections to Clients

REQUIREMENT(S): DECADE SHOULD require that network connections be made from DECADE clients to DECADE servers (i.e., not from the server to the DECADE client).

RATIONALE: Many household networks and operating systems have firewalls and NATs configured by default to block incoming connections. To ease deployment by avoiding configuration changes and help mitigate security risks, DECADE should not

require clients to listen for any incoming network connections (beyond what is required by any other already-deployed application).

4.1.2. Security

4.1.2.1. Secure Transport

REQUIREMENT(S): DECADE MUST contain a mode in which all communication between a DECADE client and server is over a secure transport that provides confidentiality, integrity, and authentication.

RATIONALE: Target Applications may wish to write sensitive data. To satisfy this use case, DECADE must provide a mode in which all communication between a DECADE client and server occurs over a secure transport protocol (e.g., SSL/TLS).

4.1.3. Error and Failure Conditions

4.1.3.1. Overload Condition

REQUIREMENT(S): In-network storage, which is operating close to its capacity limit (e.g., too busy servicing other requests), MUST be permitted to reject requests and not be required to generate responses to additional requests. In-network storage MUST also be permitted to redirect requests (see Section 4.1.3.5) as a load-shedding technique.

RATIONALE: Forcing the in-network storage to respond to requests when operating close to its capacity can impair its ability to service existing requests, and thus is permitted to avoid generating responses to additional requests.

4.1.3.2. Insufficient Resources

REQUIREMENT(S): DECADE MUST support an error condition indicating to a DECADE client that resources (e.g., storage space) were not available to service a request (e.g., storage quota exceeded when attempting to write data).

RATIONALE: The currently-used resource levels within the in-network storage are not locally-discoverable, since the resources (disk, network interfaces, etc) are not directly attached. In order to allocate resources appropriately amongst remote clients, a client must be able to determine when resource limits have been reached. The client can then respond by explicitly freeing necessary resources or waiting for such resources to be freed.

4.1.3.3. Unavailable and Deleted Data

REQUIREMENT(S): DECADE MUST support error conditions indicating that (1) data was rejected from being written, (2) deleted, or (3) marked unavailable by a storage provider.

RATIONALE: Storage providers may require the ability to (1) avoid storing, (2) delete, or (3) quarantine certain data that has been identified as illegal (or otherwise prohibited). DECADE does not indicate how such data is identified, but applications using DECADE should not break if a storage provider is obligated to enforce such policies. Appropriate error conditions should be indicated to applications.

4.1.3.4. Insufficient Permissions

REQUIREMENT(S): DECADE MUST support error conditions indicating that the requesting client does not have sufficient permissions to access requested data objects.

RATIONALE: DECADE clients may request objects to which they do not have sufficent access permissions, and DECADE servers must be able to signal that this has occurred. Note that access permissions may be insufficient due to a combination of the credentials presented by a client as well as additional policies defined by the storage provider.

4.1.3.5. Redirection

REQUIREMENT(S): DECADE SHOULD support the ability for a DECADE server to redirect requests to another DECADE server. This may either be in response to an error, failure, or overload condition, or to support capabilities such as load balancing.

RATIONALE: A DECADE server may opt to redirect requests to another server to support capabilities such as load balancing, or if the implementation decides that another DECADE server is in a better position to handle the request due to either its location in the network, server status, or other consideration.

4.2. Transfer and Latency Requirements

4.2.1. Low-Latency Access

REQUIREMENT(S): DECADE SHOULD provide "low-latency" access for application clients. DECADE MUST allow clients to specify at least two classes of services: lowest possible latency and latency non-critical.

RATIONALE: Some applications may have requirements on delivery time (e.g., live streaming [PPLive]). The user experience may be unsatisfactory if the use of in-network storage results in lower performance than connecting directly to remote clients over a low-speed, possibly congested uplink. Additionally, the overhead required for control-plane operations in DECADE must not cause the latency to be higher than for a low-speed, possibly congested uplink. While it is impossible to make a guarantee that a system using in-network storage will always outperform a system that does not for every transfer, the overall performance of the system should be improved compared with direct low-speed uplink connections, even considering control overhead.

4.2.2. Data Object Size

REQUIREMENT(S): DECADE MUST allow for efficient data transfer of small objects (e.g., 16KB) between a DECADE client and in-network storage with minimal additional latency imposed by the protocol.

RATIONALE: Though Target Applications are frequently used to share large amounts of data (e.g., continuous streams or large files), the data itself is typically subdivided into smaller chunks that are transferred between clients. Additionally, the small chunks may have requirements on delivery time (e.g., in a live-streaming application). DECADE must enable data to be efficiently transferred amongst clients at this granularity. It is important to note that DECADE may be used to store and retrieve larger objects, but protocol(s) should not be designed such that usage with smaller data objects cannot meet the requirements of Target Applications.

4.2.3. Communication among DECADE Servers

REQUIREMENT(S): DECADE SHOULD support the ability for two in-network storage elements in different administrative domains to write and/or read data directly between each other (if authorized as described in Section 4.7). If such a capability is supported, this MAY be the same (or a subset or extension of) as the DECADE protocol(s) used by clients to access data.

RATIONALE: Allowing server-to-server communication can reduce latency in some common scenarios. Consider a scenario when a DECADE client is downloading data into its own storage from another client's in-network storage. One possibility is for the client to first download the data itself, and then upload it to its own storage. However, this uploading causes unnecessary latency and network traffic. Allowing the data to be downloaded from the remote in-network storage into the client's own in-network storage can alleviate both.

4.3. Data Access Requirements

4.3.1. Reading/Writing Own Storage

REQUIREMENT(S): DECADE MUST support the ability for a DECADE client to read data from and write data to its own in-network storage.

RATIONALE: Two basic capabilities for any storage system are reading and writing data. A DECADE client can read data from and write data to in-network storage space that it owns.

4.3.2. Access by Other Users

REQUIREMENT(S): DECADE MUST support the ability for a user to apply access control policies to users other than itself for its storage. The users with whom access is being shared can be under a different administrative domain than the user who owns the in-network storage. The authorized users may read from or write to the user's storage.

RATIONALE: Endpoints in Target Applications may be located across multiple ISPs under multiple administrative domains. Thus, to be useful by Target Applications, DECADE allows a user to implement access control policies for users that may or may not be known to the user's storage provider.

4.3.3. Negotiable Data Transport Protocol

REQUIREMENT(S): DECADE MUST support the ability for a DECADE client to negotiate with its in-network storage about which protocol it can use to read data from and write data to its In-network storage. DECADE MUST specify at least one mandatory protocol to be supported by implementations; usage of a different protocol may be selected via negotiation.

RATIONALE: Since typical data transport protocols (e.g., NFS and WebDAV) already provide read and write operations for network storage, it may not be necessary for DECADE to define such operations in a new protocol. However, because of the particular application requirements and deployment considerations, different applications may support different protocols. Thus, a DECADE client must be able to select an appropriate protocol also supported by the in-network storage. This requirement also follows as a result of the requirement of Separation of Control and Data Operations (Section 4.3.4).

4.3.4. Separation of Data and Control Policies

REQUIREMENT(S): DECADE Protocol(s) MUST only provide a minimal set of core operations to support diverse policies implemented and desired by Target Applications.

RATIONALE: Target Applications support many complex behaviors and diverse policies to control and distribute data, such as (e.g., search, index, setting permissions/passing authorization tokens). Thus, to support such Target Applications, these behaviors must be logically separated from the data transfer operations (e.g., read, write). Some minimal overlap (for example obtaining credentials needed to encrypt or authorize data transfer using control operations) may be required to be directly specified by DECADE.

4.4. Data Management Requirements

4.4.1. Agnostic of reliability

REQUIREMENT(S): DECADE SHOULD remain agnostic of reliability/ fault-tolerance level offered by storage provider.

RATIONALE: Providers of a DECADE service may wish to offer varying levels of service for different applications/users. However, a single compliant DECADE client should be able to use multiple DECADE services with differing levels of service.

4.4.2. Data Object Attributes

REQUIREMENT(S): DECADE MUST support the ability to associate attributes with data objects with a scope local to a DECADE server, and for DECADE clients to query these attributes. DECADE protocol(s) MUST transmit any attributes using an operating system-independent and architecture-independent standard format. DECADE protocol(s) MUST be designed such that new attributes can be added by later protocol revisions or extensions.

RATIONALE: DECADE supports associating attributes (e.g., a time-to-live, creation timestamp, or object size) with a data object. These attributes are local to a data object stored by a particular DECADE server, and are thus not applied to any DECADE server or client to which the data object is copied. These attributes may be used as hints to the storage system, internal optimizations, or as additional information queryable by DECADE clients.

4.4.3. Time-to-live for Written Data Objects

REQUIREMENT(S): DECADE MUST support the ability for a DECADE client to indicate a time-to-live value (or expiration time) indicating a length of time until particular data can be deleted by the in-network storage element.

RATIONALE: Some data objects written by a DECADE client may be usable only within a certain window of time, such as in live-streaming P2P applications. Providing a time-to-live value for data objects (e.g., at the time they are written) can reduce management overhead by avoiding many 'delete' commands sent to in-network storage. The in-network storage may still keep the data in cache for bandwidth optimization. But this is guided by the privacy policy of the DECADE provider.

4.4.4. Offline Usage

REQUIREMENT(S): DECADE MAY support the ability for a user to provide authorized access to its in-network storage even if the user has no DECADE applications actively running or connected to the network.

RATIONALE: If an application desires, it can authorize remote clients to access its storage even after the application exits or network connectivity is lost. An example use case is mobile scenarios, where a client can lose and regain network connectivity very often.

4.5. Data Naming Requirements

4.5.1. Unique Names

REQUIREMENT(S): DECADE MUST support a naming scheme that ensures a high probability of uniqueness and supports the operation of DECADE servers under diverse administrative domains with no coordination. DECADE SHOULD provide a mechanism (minimally rejection) to handle the improbable case of a collision.

RATIONALE: When writing a data object, a DECADE Client should be able to write it without being concerned over whether an object of the same name already exists, unless the existing object contains the exact same data. Note that it may be reasonable for DECADE to satisfy this requirement probabilistically (e.g., using a hashing mechanism). As a result, it is wise to provide a collision handling mechanism, even if the probability of collisions is extremely low.

4.6. Resource Control

4.6.1. Multiple Applications

REQUIREMENT(S): DECADE SHOULD support the ability for users to define resource sharing policies for multiple applications (DECADE clients) being run/managed by the user.

RATIONALE: A user may own in-network storage and share the in-network storage resources amongst multiple applications. For example, the user may run one or more video-on-demand application(s) and a live-streaming application(s) which both make use of the user's in-network storage. The applications may be running on different machines and may not directly communicate. Thus, DECADE should enable the user to determine resource sharing policies between the applications.

One possibility is for a user to indicate the particular resource sharing policies between applications out-of-band (not using a standard protocol), but this requirement may manifest itself in passing values within DECADE protocol(s) to identify individual applications. Such identifiers can be either user-generated or server-generated and do not need to be registered by IANA.

4.6.2. Per-Remote-Client, Per-Data Control

REQUIREMENT(S): A DECADE client MUST be able to assign resource policies (bandwidth share, storage quota, and network connection quota) to individual remote clients for reading from and writing particular data to its in-network storage within a particular range of time. The DECADE server MUST enforce these constraints.

RATIONALE: Target Applications can rely on control of resources on a per-remote-client or per-data basis. For example, application policy may indicate that certain remote clients have a higher bandwidth share for receiving data [LLSB08]. Additionally, certain data (e.g., chunks) may be distributed with a higher priority. As another example, when allowing a remote client to write data to a user's in-network storage, the remote client may

be restricted to write only a certain amount of data. Since the client may need to manage multiple clients accessing its data, it should be able to indicate the time over which the granted resources are usable. For example, an expiration time for the access could be indicated to the server after which no resources are granted (e.g., indicate error as access denied).

4.6.3. Server Involvement

REQUIREMENT(S): A DECADE client SHOULD be able to indicate to a DECADE server, without itself contacting the server, resource control policies for remote clients' requests.

RATIONALE: One important consideration for in-network storage elements is scalability, since a single storage element may be used to support many users. Many Target Applications use small chunk sizes and frequent data exchanges. If such an application employed resource control and contacted the in-network storage element for each data exchange, this could present a scalability challenge for the server as well as additional latency for clients.

Our preferred alternative is to let requesting users obtain signed resource control policies (in the form of a token) from the owning user, and then users can then present the policy to the storage directly. This can result in reduced messaging handled by the in-network storage.

4.7. Authorization

4.7.1. Per-Remote-Client, Per-Data Read Access

REQUIREMENT(S): A DECADE Client MUST be able to control which individual remote clients are authorized to read particular data from its in-network storage.

RATIONALE: A Target Application can control certain application-level policies by sending particular data (e.g., chunks) to certain remote clients. It is important that remote clients not be able to circumvent such decisions by arbitrarily reading any data in in-network storage.

4.7.2. Per-User Write Access

REQUIREMENT(S): A DECADE Client MUST be able to control which individual remote clients are authorized to write data into its in-network storage.

RATIONALE: The space managed by a user in in-network storage can be a limited resource. At the same time, it can be useful to allow remote clients to write data directly to a user's in-network storage. Thus, a DECADE client should be able to grant only certain remote clients this privilege.

4.7.3. Default Access Permissions

REQUIREMENT(S): Unless read or write access is granted by a DECADE Client to a remote client, the default access MUST be no access.

RATIONALE: The current leading proposal for an access control model in DECADE is via token passing, resulting in a system with little state on the server side.

4.7.4. Authorization Checks

REQUIREMENT(S): In-network storage MUST check the authorization of a client before it executes a supplied request. The in-network storage MAY use optimizations to avoid such authorization checks as long as the enforced permissions are the same.

RATIONALE: Authorization granted by a DECADE client are meaningless unless unauthorized requests are denied access. Thus, the in-network storage element must verify the authorization of a particular request before it is executed.

4.7.5. Cryptographic Credentials

REQUIREMENT(S): Access MUST be able to be granted using cryptographic credentials.

RATIONALE: DECADE clients may be operating on hosts without constant network connectivity or without a permanent attachment address (e.g., mobile devices). To support access control with such hosts, DECADE servers must support access control policies that use cryptographic credentials, not simply by tying access to IP addresses.

4.7.6. Server Involvement

REQUIREMENT(S): A DECADE client SHOULD be able to indicate, without contacting the server itself, access control policies for remote clients' requests.

RATIONALE: See discussion in Section 4.6.3.

4.7.7. Protocol Reuse

REQUIREMENT(S): If possible, DECADE SHOULD reuse existing protocol and mechanisms for Authentication, Access, and Authorization (AAA).

RATIONALE: If possible, reusing an existing AAA protocol/mechanism will minimize the development required by applications, and will maximize interoperability of the DECADE protocol with existing infrastructure.

4.8. Non-Requirements

4.8.1. Application-defined Properties and Metadata

REQUIREMENT(S): DECADE MUST NOT provide a mechanism for associating Application-defined properties (metadata) with data objects beyond what is provided by Section 4.4.2.

RATIONALE: Associating key-value pairs that are defined by Target Applications with data objects introduces substantial complexity to the DECADE protocol. If Target Applications wish to associate additional metadata with a data object, possible alternatives include (1) managing such metadata within the Target Application itself, (2) storing metadata inside of the data object, or (3) storing metadata in a different data object at the DECADE server.

5. Storage Requirements

This section details the requirements of the underlying storage used to support the DECADE protocol(s).

5.1. Immutable Data

REQUIREMENT(S): DECADE MUST only store and manage data objects that are immutable once they are written to storage.

RATIONALE: Immutable data objects are an important simplification in DECADE. Reasonable consistency models for updating existing objects would significantly complicate implementation (especially if implementation chooses to replicate data across multiple

servers). If a user needs to update a resource, it can write a new resource and then distribute the new resource instead of the old one.

5.2. Explicit Deletion of Data

REQUIREMENT(S): DECADE MUST support the ability for a DECADE client to explicitly delete data from its own in-network storage.

RATIONALE: A DECADE client may continually be writing data to its in-network storage. Since there may be a limit (e.g., imposed by the storage provider) to how much total storage can be used, some data may need to be removed to make room for additional data. A DECADE client should be able to explicitly remove particular data. This may be implemented using existing protocols.

5.3. Multiple writing

REQUIREMENT(S): DECADE MUST NOT allow multiple simultaneous writers for the same object. Implementations MUST raise an error to one of the writers.

RATIONALE: This avoids data corruption in a simple way while remaining efficient. Alternately, the DECADE server would need to either manage locking, handle write collisions, or merge data, all of which reduce efficiency and increase complexity.

5.4. Multiple reading

REQUIREMENT(S): DECADE MUST allow for multiple simultaneous readers for an object.

RATIONALE: One characteristic of Target Applications is the ability to upload an object to multiple clients. Thus, it is natural for DECADE to allow multiple readers to read the content concurrently.

5.5. Reading before completely written

REQUIREMENT(S): DECADE MAY allow readers to read from objects before they have been completely written.

RATIONALE: Some Target Applications (in particular, P2P streaming) can be sensitive to latency. A technique to reduce latency is to remove store-and-forward delays for data objects (e.g., make the object available before it is completely written). Appropriate handling for error conditions (e.g., a disappearing writer) needs to be specified.

5.6. Hints concerning usage of written data

REQUIREMENT(S): DECADE MAY allow writers of new objects to indicate specific hints concerning how the objects are expected to be used (e.g., access frequency or time-to-live).

RATIONALE: Different Target Applications may have different usage patterns for objects written to in-network storage. For example, a P2P live streaming application may indicate to a DECADE server that the objects are expected to have a short time-to-live, but read frequently. The DECADE server may then opt to persist the objects in memory instead of in disk.

5.7. Writing model

REQUIREMENT(S): DECADE storage MUST provide at least a writing model (while writing an object) that appends data to data already written.

RATIONALE: Depending on the object size (e.g., chunk size) used by a Target Application, the application may need to send data to the DECADE server in multiple packets. To keep implementation simple, the DECADE must at least support the ability to write the data sequentially in the order received. Implementations MAY allow application to write data in an object out-of-order (but MUST NOT overwrite ranges of the object that have already been written).

5.8. Storage Status

REQUIREMENT(S): A DECADE client MUST be able to read current resource usage (including list of written data objects), resource quotas, and access permissions for its in-network storage. The returned information MUST include resource usage resulting from the client’s own usage and usage by other clients that have been authorized to read/write objects or open connections to that client’s storage. DECADE protocol(s) MUST support the ability for a DECADE client to read aggregated resource usage information (across all other clients to which it has authorized access), and MAY support the ability to request this information for each other authorized client.

RATIONALE: The resources used by a client are not directly-attached (e.g., disk, network interface, etc). Thus, the client cannot locally determine how such resources are being used. Before storing and retrieving data, a client should be able to determine which data is available (e.g., after an application restart). Additionally, a client should be able to determine resource

availability to better allocate them to remote clients. Due to scalability issues, it is not required that DECADE support returning usage information broken down by each remote client which has been authorized access, but this feature may be useful in certain deployment scenarios.

6. Discovery Requirements

6.1. Requirements

6.1.1. Locating DECADE Servers

REQUIREMENT(S): The DECADE Discovery mechanism MUST allow a DECADE Client to identify one or more DECADE Servers to which it is authorized to read/write data and to which it may authorize other DECADE Clients to read/write data, or fail if no such DECADE Servers are available.

RATIONALE: A basic goal of DECADE is to allow DECADE Clients to read/write data for access by other DECADE Clients or itself. Executing the Discovery process should result in a DECADE Client finding a DECADE Server that it can use for these purposes. It is possible that no such DECADE Servers are available. Note that even if a DECADE Client may not successfully locate a DECADE Server that fits these criteria, it may still read/write data from/to a DECADE Server if authorized by another DECADE Client.

6.1.2. Support for Clients Behind NATs and Firewalls

REQUIREMENT(S): The Discovery mechanism MUST support DECADE Clients operating behind NATs and Firewalls without requiring additional network support (e.g., Application-level Gateways).

RATIONALE: NATs and Firewalls are prevalent in network deployments, and it is common for Target Applications that include a DECADE Client to be deployed behind these devices.

6.1.3. Prefer Existing Protocols

REQUIREMENT(S): The DECADE Server discovery mechanism SHOULD leverage existing mechanisms and protocols wherever possible.

RATIONALE: Existing protocols such as DNS and DHCP are widespread. Using existing protocols, or combinations of the protocols that have been specified in other contexts, is strictly preferred over developing a new discovery protocol for DECADE.

7. Future Considerations

This section enumerates considerations that should be taken into account during the DECADE design and implementation. They have been intentionally omitted as requirements since they are either out of scope or implementation-dependent. Nevertheless, enumerating them may help to guide future application of the requirements included in this document.

7.1. Fairness

To provide fairness among users, the in-network storage provider should assign resource (e.g., storage, bandwidth, connections) quota for users. This can prevent a small number of clients from occupying large amounts of resources on the in-network storage, while others starve.

7.2. Removal of Duplicate Data Objects

There are actually two possible scenarios. The first is the case of removing duplicates within one particular DECADE server (or logical server). While not a requirement, as it does not impact the protocol, a DECADE server may implement internal mechanisms to monitor for duplicate objects and use internal mechanisms to prevent duplication in internal storage.

The second scenario is removing duplicates across a distributed set of DECADE servers. DECADE does not explicitly design for this, but does offer a redirection mechanism (Section 4.1.3.5) that is one primitive that may be used to support this feature in certain deployment scenarios.

7.3. Gaming of the Resource Control Mechanism

The particular resource control policy communicated by a DECADE protocol and enforced by the scheduling system of a DECADE implementation may be open to certain gaming by clients. for example by specifying many small peers to increase total throughput or inciting overload conditions at a DECADE server. Identifying and protecting against all such opportunities for gaming is outside the scope of this document, but DECADE protocol(s) and implementations SHOULD be aware that opportunities to game the system may be attempted.

8. Security Considerations

The security model is an important component of DECADE. It is

crucial for users to be able to manage and limit distribution of content to only authorized parties, and the mechanism needs to work on the general Internet which spans multiple administrative and security domains. Previous sections have enumerated detailed requirements, but this section discusses the overall approach and other considerations that do not merit requirements.

8.1. Authentication and Authorization

DECADE only uses authentication when allowing a particular client to access its own storage at a server. DECADE servers themselves do not authenticate other clients which are reading/writing a client's own storage. Instead, DECADE relies on clients to authenticate others to access its storage, and then communicate the result of that authentication to the DECADE server so that the DECADE server may implement the necessary authorization checks.

8.2. Encrypted Data

DECADE Servers provide the ability to write raw data objects (subject to any policies instituted by the owner/administrator of the DECADE server, which are outside of the scope of this document). Thus, DECADE clients may opt to encrypt data before it is written to the DECADE Server. However, DECADE itself does not provide encryption of data objects other than is provided by Section 4.1.2.1.

9. IANA Considerations

There are no IANA considerations with this document.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[I-D.ietf-decade-problem-statement]
Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", draft-ietf-decade-problem-statement-03 (work in progress), March 2011.

[I-D.ietf-decade-arch]

Alimi, R., Yang, Y., Rahman, A., Kutscher, D., and H. Liu, "DECADE Architecture", draft-ietf-decade-arch-02 (work in progress), July 2011.

[LLSB08] Levin, D., LaCurts, K., Spring, N., and B. Bhattacharjee, "BitTorrent is an Auction: Analyzing and Improving BitTorrent's Incentives", SIGCOMM 2008, August 2008.

[PPLive] "PPLive", <http://www.pplive.com>.

Appendix A. Acknowledgments

We would also like to thank Haibin Song for substantial contributions to earlier versions of this document. We would also like to thank Reinaldo Penno, Alexey Melnikov, Rich Woundy, Ning Zong, Roni Even, David McDysan, Borje Ohlman, Dirk Kutscher, Akbar Rahman, Xiao Zhu, Yunfei Zhang, and Jin Peng for contributions and general feedback.

Authors' Addresses

Yingjie Gu
Huawei
No. 101 Software Avenue
Nanjing, Jiangsu Province 210012
P.R.China

Phone: +86-25-56624760
Email: guyingjie@huawei.com

David A. Bryan
Polycom, Inc.

Email: dbryan@ethernot.org

Yang Richard Yang
Yale University

Email: yry@cs.yale.edu

Richard Alimi
Google

Email: ralimi@google.com

DECADE
Internet-Draft
Intended status: Informational
Expires: February 14, 2013

Y. Gu
Huawei
D. Bryan
Ethernot.org
Y. Yang
Yale University
P. Zhang
Tsinghua University/Yale University
R. Alimi
Google

August 13, 2012

DECADE Requirements
draft-ietf-decade-reqs-08

Abstract

The target of the DECoupled Application Data Enroute (DECADE) system is to provide an open and standard in-network storage system for applications, primarily P2P (peer-to-peer) applications, to store, retrieve and manage their data. This draft enumerates and explains requirements, not only for storage and retrieval, but also for data management, access control and resource control, that should be considered during the design and implementation of a DECADE-compatible system. These are requirements on the entire system; some of the requirements may eventually be implemented by an existing protocol with/without some extensions (e.g., a protocol used to read and write data from the storage system). The requirements in this document are intended to ensure that a DECADE-compatible system architecture includes all of the desired functionality for intended applications.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-

Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 14, 2013.

Table of Contents

1. Introduction 5
2. Terminology 6
2.1. DECADE-compatible Client 6
2.2. DECADE-compatible Server 6
2.3. DECADE Storage Provider 6
2.4. DECADE Account 6
2.5. Resource Provider 6
2.6. Resource Consumer 6
2.7. Content Distribution Application 6
2.8. Target Application 7
2.9. Application End-Point 7
2.10. Data Object 7
2.11. DECADE-compatible System 7
2.12. DECADE Resource Protocol (DRP) Functions 7
2.13. DECADE Standard Data Transfer Protocol (SDT) Functions . . 7
3. System and Protocol Components 8
4. Requirements Structure 9
5. Data Object Requirements 9
5.1. Data Name Uniqueness 9
5.2. Verifiable Name-Object Binding 10
5.3. Data Object Size 10
5.4. Data Object Attributes 10
5.5. Application-defined Object Properties and Metadata 11
6. Access and Authorization Requirements 11
6.1. Provider Access 11
6.2. Secure Authorization 11
6.3. Consumer Access 12
6.4. Provider Authorization When Offline 12
6.5. Local Authorization 12
6.6. Access Control Granularity 12
6.7. Default Access Permissions 12
6.8. Connectivity Supporting NAT and Firewall Traversal 13
6.9. DECADE Server Discovery 13
7. Data Transfer Requirements 13
7.1. Negotiable Standard Data Transport Protocol 13
7.2. Atomic or Partial Read/Write 14
7.3. Secure Data Transport 14
7.4. Concurrent Read 14
7.5. Concurrent Write 14
7.6. Read Before Write Complete 15
7.7. Redirection of Transport 15
8. Resource Control Requirements 16
8.1. Multiple Applications Sharing Resources 16
8.2. Per-Client Resource Policy 16
8.3. Distributed Resource Sharing Specification 16
8.4. Resource Set 17

9. Error and Failure Handling Requirements 17
9.1. Illegal Data Object 17
9.2. Invalid Access Authorization 18
9.3. Insufficient Resources 18
9.4. Overload Condition 18
9.5. Attack Mitigation 19
10. Management Info Requirements 19
10.1. Account Status 19
11. Security Considerations 19
11.1. Authentication and Authorization 20
11.2. Confidentiality 20
11.3. Attack Mitigation 20
12. IANA Considerations 20
13. References 20
13.1. Normative References 20
13.2. Informative References 20
Appendix A. Acknowledgments 21
Authors' Addresses 21

1. Introduction

The object of the DECoupled Application Data Enroute (DECADE) system is to provide an open and standard in-network storage for content distribution applications, where data is typically broken into one or more chunks and then distributed. This may already include many types of applications including P2P applications, IPTV (Internet Protocol Television), and VoD (Video on Demand). (For a precise definition of the applications targeted in DECADE system, see the definition for Target Application in Section 2.) Instead of always transferring data directly from a source/owner client to a requesting client, the source/owner client can write to and manage its content on its in-network storage. The requesting client can get the address of the in-network storage pertaining to the source/owner client and read data from the storage.

This draft enumerates and explains the rationale behind specific requirements on the protocol design and on any data store implementation that may be used to implement DECADE servers that should be considered during the design and implementation of a DECADE-compatible system. As such, it does not include general guiding principles. General design considerations, explanation of the problem being addressed, and enumeration of the types of applications to which a DECADE-compatible system may be suited is not considered in this document. For general information, please see [RFC6646] and [I-D.ietf-decade-arch].

This document enumerates the requirements to enable target applications to utilize in-network storage. In this context, using storage resources includes not only basic capabilities such as writing, reading, and managing data, but also controlling access for particular remote clients with which it is sharing data. Additionally, we also consider controlling the resources used by remote clients when they access data as an integral part of utilizing the network storage.

This document discusses requirements pertaining to DECADE-compatible protocol(s). In certain deployments, several logical in-network storage systems could be deployed (e.g., within the same administrative domain). These in-network storage systems can communicate and transfer data through internal or non-standard communication messages that are outside of the scope of these requirements, but they should use DECADE-compatible protocol(s) when communicating with other DECADE-compatible in-network storage systems.

2. Terminology

This document uses the term 'In-network storage' which is defined in [RFC6646].

This document also defines these additional terms:

2.1. DECADE-compatible Client

A DECADE-compatible client uploads and/or retrieves data from DECADE-compatible servers. We use the shorter term "client" if there is no ambiguity.

2.2. DECADE-compatible Server

A DECADE-compatible server stores data inside the network, and thereafter manages both the stored data and access to those data. We use the shorter term "server" if there is no ambiguity.

2.3. DECADE Storage Provider

A DECADE Storage Provider, or Storage Provider for short, deploys and/or manages DECADE-compatible server(s) within a network.

2.4. DECADE Account

An account of a DECADE-compatible server has associated cryptographic credentials for access control, and resource allocation attributes on the server.

2.5. Resource Provider

A client which has the account cryptographic credentials of a DECADE account at a DECADE-compatible server. We use the short term "Provider" if there is no ambiguity.

2.6. Resource Consumer

A client which tries to access a DECADE account but does not have the account's cryptographic credentials. We use the short term "Consumer" if there is no ambiguity.

2.7. Content Distribution Application

A Content Distribution Application is an application (e.g., P2P, traditional CDN, or hybrid P2P/CDN) designed for dissemination of a large amounts of content (e.g., files, or video streams) to multiple content consumers. Content Distribution Applications may divide

content into smaller blocks for dissemination.

2.8. Target Application

An application with that includes a DECADE-compatible client along with other application functionalities to explicitly control the usage of resources of DECADE-compatible servers to deliver content to other users. A primary type of Target Application we consider is Content Distribution Applications. A Target Application divides content into smaller blocks for more flexible distribution (e.g., over multiple application-level paths). We use the term Target Application to refer to the type of applications that are explicitly (but not exclusively) supported by DECADE system.

2.9. Application End-Point

An Application End-Point is an instance of a Target Application. A particular Application End-Point might be a Provider, a Consumer, or both. For example, an Application End-Point might be an instance of a video streaming client, or it might be the source providing the video to a set of clients.

2.10. Data Object

A data object is the unit of data stored and retrieved from a DECADE-compatible server. The data object is a string of raw bytes. The server maintains metadata associated with each data object, but the metadata is not included in the data object.

2.11. DECADE-compatible System

A system which is composed of DECADE-compatible clients, DECADE-compatible servers and in-network storage. A DECADE-compatible system MUST obey all the requirements defined in this document.

2.12. DECADE Resource Protocol (DRP) Functions

A set of functions for communication of access control and resource scheduling policies from a DECADE client to a server, as well as between servers.

2.13. DECADE Standard Data Transfer Protocol (SDT) Functions

A set of functions to be used to transfer data objects to and from a DECADE server.

3. System and Protocol Components

To organize requirements, we consider that a DECADE-compatible system consists of two logical sets of functions, as shown in Figure 1. The first set of functions, which we refer to as the DECADE Resource Protocol (DRP) functions, is responsible for communication of access control and resource scheduling policies from a client to a server, as well as between servers. A DECADE-compatible system will include exactly one DRP for interoperability and a common format through which these policies can be communicated.

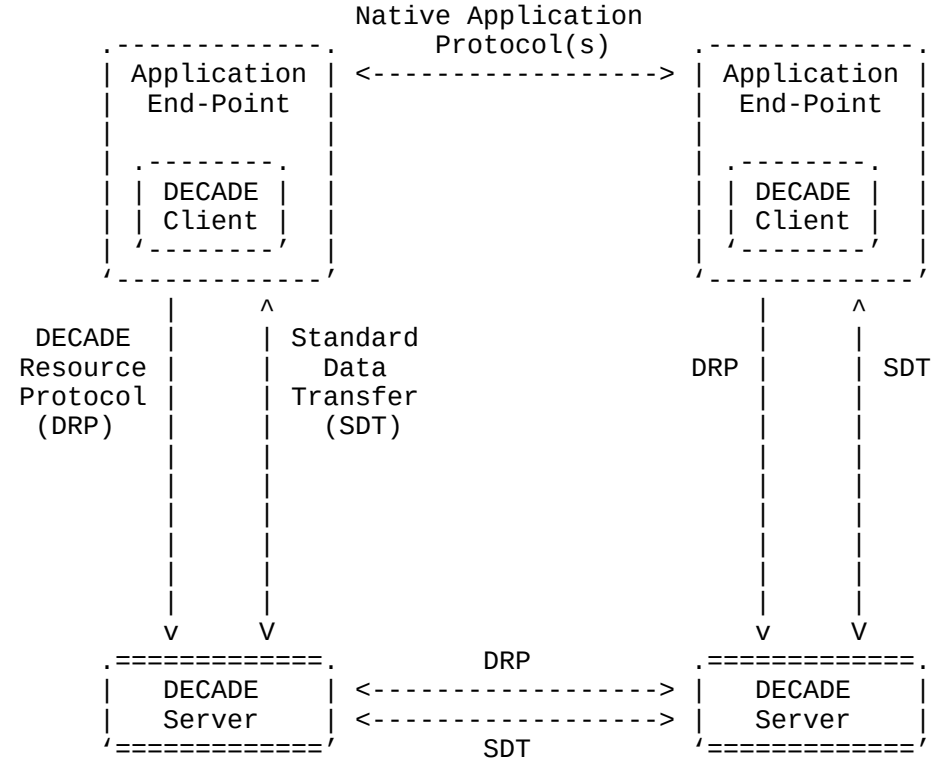


Figure 1: Protocol Components and Generic Flow

Second, the second set of functions, referred to as the Standard Data Transfer (SDT) functions, will be used to transfer data objects to and from a server. A DECADE-compatible system may support multiple SDT's. If there are multiple SDT's, a negotiation mechanism will be used to determine a suitable SDT between the client and server.

The two sets of functions (DRP and SDT) will be either separate or combined on the wire. If they are combined, DRP messages can be

piggy-backed within some extension fields provided by certain SDT protocols. In such a scenario, DRP is technically a data structure (transported by other protocols), but it can still be considered as a logical protocol that provides the services of configuring DECADE-compatible resource usage. If the protocols are physically separate on the wire, DRP can take the form of a separate control connection open between the a DECADE-compatible client and server. Hence, this document considers SDT and DRP as two separate, logical functional components for clarity.

4. Requirements Structure

This document specifies the requirements for the DECADE DRP and SDT functions, either existing ones or new ones, and storage system to enable Target Applications to make use of storage within the network, leaving specific storage system considerations to the implementation of the storage servers as much as possible. For this reason, we consider two primary categories of requirements:

- Protocol Requirements: Protocol requirements for Target Applications to make use of in-network storage within their own data dissemination schemes. Development of these requirements is guided by a study of data access, search and management capabilities used by Target Applications. These requirements may be met by a combination of existing protocols and new protocols.

- Storage Requirements: Functional requirements necessary for the back-end storage system employed by the DECADE server. Development of these requirements is guided by a study of the data access patterns used by Target Applications. These requirements should be met by the underlying data transport used by DECADE system. In this document, we use "data transport" to refer to a protocol used to read and write data from in-network storage.

This specification discusses the requirements of functionality implemented with a storage system and within applications, to permit interoperable communications concerning the manipulation of stored content.

5. Data Object Requirements

5.1. Data Name Uniqueness

REQUIREMENT(S): Each Data Object should be named to allow access. DECADE-compatible protocol(s) MUST support a data object naming scheme that ensures a high probability of uniqueness, with no coordination among multiple Storage Providers. In other words, two Data Objects with the same name should be the same content with high probability. A DECADE-compatible server SHOULD be able to respond to a DECADE-compatible client with an error indicating potential name conflicts.

RATIONALE: Although the intention of unique names is to avoid name collisions, it does not have to be an absolutely zero possibility. Hence, it is required to provide a collision handling mechanism.

EXCEPTION: While a DECADE-compatible server is overloaded or consider a request as an attack, it does not to generate a response to indicate name collisions.

5.2. Verifiable Name-Object Binding

5.3. Data Object Size

REQUIREMENT(S): DECADE MUST allow for efficient storage and data transfer of small data objects (e.g., 16KB) without large control overhead.

RATIONALE: Though Target Applications are frequently used to share large amounts of data (e.g., continuous streams or large files), the data itself is typically subdivided into smaller data objects (chunks) for flexibility (e.g., reliability and multi-path transmission).

5.4. Data Object Attributes

REQUIREMENT(S): DECADE MUST support the ability to associate a set of system attributes with a data object with a scope local to a DECADE-compatible server. In particular, the set MUST include time-to-live (or expiration time), creation timestamp, object size, and object type. A DECADE-compatible client, with access permission, MUST be able to query the set of system attributes. The transmission of the attributes MUST use an operating system-independent and architecture-independent standard format. An ability to extend the set of attributes MUST exist.

RATIONALE: The values of attributes associated with a data object are local to a particular DECADE-compatible server. These attributes may be used as hints to the storage system, internal optimizations, or as additional information query-able by DECADE-

compatible clients. The particular requirement for including time-to-live (TTL) is that a data object written by a DECADE-compatible client may be usable only within a certain window of time, such as in a live-streaming P2P application. Providing a time-to-live value for a data object (e.g., at the time it is written) can reduce management overhead by avoiding many 'delete' commands sent to DECADE-compatible server. The server may still retain a data object for bandwidth optimization, but this should be guided by the privacy policy of the DECADE Storage Provider.

5.5. Application-defined Object Properties and Metadata

REQUIREMENT(S): DECADE-compatible clients and DECADE-compatible servers MUST NOT be able to associate Application-defined properties (metadata) with data objects beyond what is provided by Section 5.4.

RATIONALE: Associating key-value pairs that are defined by Target Applications with data objects introduces substantial complexity. If Target Applications wish to associate additional metadata with a data object, possible alternatives include (1) managing such metadata within the Target Application itself, (2) storing metadata inside the data object, or (3) storing metadata in a different data object at the DECADE-compatible server.

6. Access and Authorization Requirements

6.1. Provider Access

REQUIREMENT(S): A Provider MUST be able to access the resources of its account.

RATIONALE: After a DECADE-compatible client owns an account on a DECADE-compatible server, it should be able to read data from and write data to the server.

6.2. Secure Authorization

REQUIREMENT(S): Access to an account on a server MUST be granted to a provider based on cryptographic security.

RATIONALE: DECADE-compatible clients may be operating on hosts without constant network connectivity or without a permanent attachment address (e.g., mobile devices). To support access control with such hosts, DECADE-compatible servers must support access control policies that use cryptographic credentials, not simply by tying access to IP addresses.

6.3. Consumer Access

REQUIREMENT(S): A Provider MUST be able to indicate to its server on whether a Consumer can access its subscribed resources.

RATIONALE: Endpoints in Target Applications may choose different servers. Thus, to be useful by Target Applications, a DECADE-compatible client must be able to specify policies on whether other DECADE-compatible clients can access its resources. The other clients may or may not be known to the server.

6.4. Provider Authorization When Offline

REQUIREMENT(S): A Provider MUST be able to grant access to a Consumer even if the Provider is not actively running or connected to its DECADE-compatible server.

RATIONALE: If an application desires, it can authorize other clients to access its storage even after the application exits or network connectivity is lost. An example use case is mobile scenarios, where a client can lose and regain network connectivity often.

6.5. Local Authorization

REQUIREMENT(S): A Provider MUST be able to indicate, without contacting its server, access control policies for Consumers. DECADE-compatible server MUST be able to authenticate the access control policies in this situation.

RATIONALE: This requirement is related with the preceding requirement, but in a perspective (i.e., protocol design). See discussions in Section 8.3.

6.6. Access Control Granularity

REQUIREMENT(S): A Provider MUST be able to control which individual clients are authorized to read/write which particular data objects from/to its in-network storage.

RATIONALE: A Target Application should able to conduct access control on the granularity of individual clients, individual data objects.

6.7. Default Access Permissions

REQUIREMENT(S): Unless read or write access is granted by a Provider, the default permission MUST be no access.

RATIONALE: This requirement is to protect client privacy by default.

6.8. Connectivity Supporting NAT and Firewall Traversal

REQUIREMENT(S): A client that is authorized to access a server MUST be supported to conduct NAT (Network Address Translation) and firewall traversal. In particular, network connections between a DECADE-compatible client and a DECADE-compatible server MUST be initiated by the client (i.e., the server must not initiate a connection to the client).

RATIONALE: Firewalls and NATs are widely used in the Internet today, both in ISP (Internet Service Provider) and Enterprise networks and by consumers. Many firewalls and NATs are configured by default to block incoming connections, which helps to mitigate security risks. Deployment of a DECADE-compatible system must not require manual modifications to such devices. This requirement applies to both potential new protocol that may be developed by the DECADE Working Group and any data transport used with DECADE protocol.

6.9. DECADE Server Discovery

REQUIREMENT(S): A mechanism for a Provider to discover and connect to its assigned server MUST be supported. The discovery SHOULD leverage existing mechanisms and protocols wherever possible. No new discovery mechanism will be defined unless there is enough evidence that no existing mechanism can work.

RATIONALE: Existing protocols such as DNS and DHCP are widespread. Using existing protocols, or combinations of the protocols that have been specified in other contexts, is strictly preferred over developing a new discovery protocol.

7. Data Transfer Requirements

7.1. Negotiable Standard Data Transport Protocol

REQUIREMENT(S): A DECADE-compatible client MUST be able to negotiate with a DECADE-compatible server about which protocol it can use to read data from and write data. DECADE MUST specify at least one mandatory transport protocol to be supported by implementations; usage of a different protocol may be selected via negotiation.

RATIONALE: Since typical data transport protocols (e.g., NFS and WebDAV) already provide read and write operations for network storage, it may not be necessary to define such operations in a new DECADE protocol. However, because of the particular application requirements and deployment considerations, different applications may support different protocols. Thus, a DECADE client must be able to select an appropriate protocol also supported by the in-network storage.

7.2. Atomic or Partial Read/Write

REQUIREMENT(S): A DECADE-compatible server MUST support the ability to read/write a complete data object in one request. It MAY support range read/write.

RATIONALE: Depending on the object size (e.g., chunk size) used by a Target Application, the application may need to send data to the DECADE-compatible server in multiple round.

7.3. Secure Data Transport

REQUIREMENT(S): A secure transport MUST be implemented for all communications between a DECADE-compatible client and DECADE-compatible server.

RATIONALE: Target Applications may wish to write sensitive data. To satisfy this use case, the communication between a DECADE-compatible client and DECADE-compatible server must be transported over a secure transport protocol (e.g., SSL/TLS).

7.4. Concurrent Read

REQUIREMENT(S): A DECADE-compatible server MUST allow for multiple simultaneous readers for a data object.

RATIONALE: One characteristic of Target Applications is the ability to upload an object to multiple clients. Thus, it is natural for DECADE-compatible server to allow multiple readers to read the same object concurrently.

7.5. Concurrent Write

REQUIREMENT(S): A DECADE-compatible server MUST NOT allow multiple simultaneous writers for the same object. A DECADE-compatible server SHOULD respond to each of the writers with an error.

RATIONALE: This avoids data corruption in a simple way while remaining efficient. Alternately, the DECADE-compatible server would need to either manage locking, handle write collisions, or merge data, all of which reduce efficiency and increase complexity.

EXCEPTION: While a DECADE-compatible server is overloaded or considers a request as an attack, it does not generate a response.

7.6. Read Before Write Complete

REQUIREMENT(S): A DECADE-compatible server MAY allow readers to read a data object before it has been completely written. In case of a write error in such a case, the DECADE-compatible server SHOULD respond to the reading client with an error indicating that the write has failed.

RATIONALE: Some Target Applications (in particular, P2P streaming) can be sensitive to latency. A technique to reduce latency is to remove store-and-forward delays for data objects (e.g., make the object available before it is completely written). Appropriate handling for error conditions (e.g., a disappearing writer) needs to be specified.

EXCEPTION: While a DECADE-compatible server is overloaded or considers a request as an attack, it does not generate a response.

7.7. Redirection of Transport

REQUIREMENT(S): A DECADE-compatible server SHOULD be able to redirect requests to another DECADE-compatible server. This may either be in response to an error, failure, overload, or to support capabilities such as load balancing.

RATIONALE: A DECADE-compatible server may opt to redirect requests to another server to support capabilities such as load balancing, or if the implementation decides that another DECADE-compatible server is in a better position to handle the request due to either its location in the network, server status, or other consideration.

EXCEPTION: A DECADE-compatible server can be configured by its service provider to support or not support redirection functionality.

8. Resource Control Requirements

8.1. Multiple Applications Sharing Resources

REQUIREMENT(S): A client MUST be able to indicate to a DECADE-compatible server about resource sharing policies among multiple Target Applications being run/managed by the same client.

RATIONALE: A client owning a DECADE account may provide the account's cryptographic credentials to multiple Providers of multiple target applications. For example, the client may run one or more video-on-demand application(s) and a live-streaming application(s) which both make use of the client's in-network storage. The concurrently running applications may be running on different machines (e.g., multiple machines at the same home network) and may not directly communicate, except through user coordination

8.2. Per-Client Resource Policy

REQUIREMENT(S): A Provider MUST be able to specify resource policies (bandwidth share, storage quota, and network connection quota) to individual Consumers for reading from and writing data to its in-network storage within a particular range of time.

RATIONALE: Target Applications can rely on control of resources on a per-client basis. For example, application policy may indicate that certain remote clients have a higher bandwidth share for receiving data [LLSB08]. Additionally, bandwidth share for receiving data [LLSB08]. Additionally, certain data (e.g., chunks) may be distributed with a higher priority. As another example, when allowing a remote client to write data to a user's in-network storage, the remote client may be restricted to write less than 100MB of data in total. Since the client may need to manage multiple clients accessing its data, it should be able to indicate the time over which the granted resources are usable. For example, an expiration time for the access could be indicated to the DECADE-compatible server after which no resources are granted (e.g., indicate error as access denied).

8.3. Distributed Resource Sharing Specification

REQUIREMENT(S): A Provider MUST be able to indicate to its DECADE-compatible server, without itself contacting the server, resource control policies for a Consumer. The DECADE-compatible server MUST be able to authenticate the resource control policies.

RATIONALE: One important consideration for a DECADE-compatible server is scalability, since a single storage element may be used to support many users. Many Target Applications use small chunk sizes and frequent data exchanges. If such an application employed resource control and contacted the DECADE-compatible server for each data exchange, this could present a scalability challenge for the server as well as additional latency for clients.

The preferred way is to let requesting clients obtain signed resource control policies (in the form of a token) from the owning client, and then requesting clients can present the policy to a DECADE-compatible server directly. This can result in reduced messaging handled by the DECADE-compatible server.

8.4. Resource Set

REQUIREMENT(S): A DECADE-compatible server MUST allow specification on the following resources: storage, bandwidth, and number of connections, and MAY allow additional types of resources to be specified.

RATIONALE: The minimum set of resources need to include the most common resources.

9. Error and Failure Handling Requirements

9.1. Illegal Data Object

REQUIREMENT(S): A DECADE-compatible server SHOULD provide an error indicating that (1) data was rejected from being written, (2) deleted, or (3) marked unavailable.

RATIONALE: DECADE Storage Providers may require the ability to (1) avoid storing, (2) delete, or (3) quarantine certain data that has been identified as illegal (or otherwise prohibited). It is not specified how such data is identified, but applications employing DECADE-compatible servers should not break if a Storage Provider is obligated to enforce such policies. Appropriate error conditions should be indicated to applications.

EXCEPTION: A DECADE-compatible server should be able to be configured to suppress Illegal Data Object responses for security reasons.

9.2. Invalid Access Authorization

REQUIREMENT(S): A DECADE-compatible server SHOULD provide an error indicating that the request does not have a valid access authorization.

RATIONALE: DECADE-compatible clients may request data objects to which they do not have a valid authorization, and DECADE-compatible servers should be able to signal that this has occurred. Invalid authorization may be due to a combination of credential issues as well as additional policies defined by a Storage Provider.

EXCEPTION: A DECADE-compatible server should be able to be configured to suppress Invalid Authorization responses for security reasons.

9.3. Insufficient Resources

REQUIREMENT(S): A DECADE-compatible server SHOULD provide a response indicating to a DECADE-compatible client that resources (e.g., storage space) were not available to service its request (e.g., storage quota exceeded when attempting to write data).

RATIONALE: The Insufficient Resources response allows a client to back off, free up necessary resources or waiting for such resources to be freed.

EXCEPTION: A DECADE-compatible server may not provide such a response if doing so increases the load or due to security concerns.

9.4. Overload Condition

REQUIREMENT(S): A DECADE-compatible server, which is operating close to its capacity limit (e.g., too busy servicing other requests), MUST be permitted to reject requests and not be required to generate response to additional requests. A DECADE-compatible server MUST also be permitted to redirect requests as a load-shedding technique.

RATIONALE: The Insufficient Resources response allows a client to back off, free up necessary resources or waiting for such resources to be freed.

EXCEPTION: A DECADE-compatible server may not provide such a response if doing so increases the load or due to security concerns.

9.5. Attack Mitigation

REQUIREMENT(S): A DECADE-compatible server MUST be permitted to reject suspicious requests and not be required to generate responses (e.g., if a client's rate of requests exceeds a pre-defined threshold).

RATIONALE: Malicious clients may attempt to attack a DECADE-compatible server by specifying many chunks to increase total throughput or inciting overload conditions. A DECADE-compatible server is permitted to reject or ignore requests that are deemed suspicious according to policies set by its DECADE service provider.

10. Management Info Requirements

10.1. Account Status

REQUIREMENT(S): A Provider MUST be able to query the resource quota as well the current usage. The response from the server MUST include resource usage resulting from both the client's own usage and usage by other clients that have been authorized to read/write objects on that client's account.

RATIONALE: The resources used by a client are not necessarily directly-attached (e.g., disk, network interface, etc). Thus, the client cannot locally determine how much resources are being used. Before storing and retrieving data, a client should be able to determine which data is available (e.g., after an application restart).

11. Security Considerations

The security model is an important component of a DECADE-compatible system. It is crucial for users to be able to manage and limit distribution of content to only authorized parties, and the mechanism needs to work on the general Internet which spans multiple administrative and security domains. Previous sections have enumerated detailed requirements, but this section discusses the overall approach and other considerations that do not merit requirements.

11.1. Authentication and Authorization

A DECADE-compatible server must validate an request to access the in-network storage.

11.2. Confidentiality

DECADE-compatible Servers provide the ability to write raw data objects (subject to any policies instituted by the owner/ administrator of the Storage Provider). Thus, DECADE-compatible clients may opt to encrypt data before it is transported to the server.

11.3. Attack Mitigation

The particular resource control policy may be open to certain attacks by clients. For example by specifying many small chunks to increase total throughput or inciting overload conditions are techniques that may be used by a client.

12. IANA Considerations

There are no IANA considerations with this document.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6646] Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", RFC 6646, July 2012.

13.2. Informative References

[I-D.ietf-decade-arch] Alimi, R., Rahman, A., Kutscher, D., and Y. Yang, "DECADE Architecture", draft-ietf-decade-arch-08 (work in progress), July 2012.

[LLSB08] Levin, D., LaCurts, K., Spring, N., and B. Bhattacharjee, "BitTorrent is an Auction: Analyzing and Improving BitTorrent's Incentives", SIGCOMM 2008, August 2008.

Appendix A. Acknowledgments

We would also like to thank Haibin Song for substantial contributions to earlier versions of this document. We would also like to thank Reinaldo Penno, Alexey Melnikov, Rich Woundy, Ning Zong, Roni Even, David McDysan, Borje Ohlman, Dirk Kutscher, Akbar Rahman, Xiao Zhu, Yunfei Zhang, Peng Zhang and Jin Peng for contributions and general feedback.

Authors' Addresses

Yingjie Gu
Huawei
No. 101 Software Avenue
Nanjing, Jiangsu Province 210012
P.R.China

Phone: +86-25-56624760
Email: guyingjie@huawei.com

David A. Bryan
Ethernot.org

Email: dbryan@ethernot.org

Yang Richard Yang
Yale University

Email: yry@cs.yale.edu

Peng Zhang
Tsinghua University/Yale University

Email: p.zhang@yale.edu

Richard Alimi
Google

Email: ralimi@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2012

D. Kutscher
M. Stiemerling
J. Seedorf
NEC

October 24, 2011

Design Considerations for a DECADE SDT
draft-kutscher-decade-protocol-00

Abstract

This memo provides some considerations for the design of a specific DECADE protocol.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 26, 2012.

Copyright Notice

Table of Contents

1. Introduction 3
2. Conceptual DECADE Protocols 3
3. Object Naming and Addressing 4
3.1. Object Naming 4
3.2. Object Addressing 6
4. Authentication and Access Control 7
5. General SDT Considerations 8
5.1. Dealing with Application Contexts, Resource Collection and Other Structure 8
5.2. Server-to-Server Communication 9
5.3. Recommendations 9
6. CDMI Considerations 9
6.1. CDMI Content Type Operations 10
6.2. CDMI Features and SDT 10
6.3. CDMI Containers 11
6.4. Object Identifiers in CDMI 12
6.5. Recommendations for SDT over CDMI 12
7. Security Considerations 13
8. Normative References 13
Appendix A. Acknowledgments 14
Authors' Addresses 14

1. Introduction

The DECADE architecture specification [refs.decadearch] describes fundamental principles for DECADE (naming, transport, authorization) and identifies a set of core components and conceptual protocols for accessing in-network storage.

A few candidate technologies have been proposed for a concrete protocol specification, such as HTTP-based protocols [RFC2616], WEBDAV [RFC3744], and CDMI [refs.CDMI] for the actual transport/ application layer functionality, as well as the NI URI scheme [refs.ni-core] for an URI format, and OAuth [RFC5849] for an authentication mechanism.

This memo is intended to aid the discussion about how to design DECADE protocols, and how to leverage existing solutions best. It further gives recommendation for a future Standard Data Transport (SDT). These recommendations are labelled with REC_XY, where XY is a sequential number.

[[Text in double square brackets (like this) is commentary.]]

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. [RFC2119]

2. Conceptual DECADE Protocols

As described in draft-ietf-decade-arch [refs.decadearch], DECADE conceptually consists of two functional building blocks: DRP (DECADE Resource Control Protocol) and SDT (Standard Data Transport).

DRP would provide conveying authorization-relevant information to servers for access control functions. As such, it is not intended as a stand-alone protocol but rather as a scheme that would be used by SDT instantiations, e.g., for passing authorization tokens from an Application Endpoint to a DECADE server. For a concrete specification, a scheme is needed that supports the representation of authorization information. That scheme should be compatible to the SDT instantiations that are specified (and envisioned to be specified). The assumption is that there would be exactly one DRP.

SDT is an actual protocol that Application Endpoints use for communicating with a DECADE server. Furthermore, SDT can also be used for server-to-server communication, i.e., when DECADE servers want to distribute content to other DECADE servers. A DECADE SDT would use an existing transport protocol and possibly an existing

application layer protocol such as HTTP or NFS. In fact, the conceptual DECADE SDT interactions that are defined in draft-decade-arch would most likely be mapped to messages, services etc. of such existing protocols, e.g., the SDT GET request would map to a HTTP GET request. The assumption is that there can be different DECADE SDT specifications, i.e., leveraging different underlying protocols. However it is also the assumption that there would be one mandatory SDT.

REC_01: The selected DRP scheme should be compatible to the different envisioned SDT instantiations.

REC_02: There should be one mandatory SDT implementation.

Figure 1: Recommendations

3. Object Naming and Addressing

3.1. Object Naming

draft-ietf-decade-arch [refs.decadearch] outlines requirements and concepts for naming DECADE resources. In essence, a DECADE name should be globally unique (with a high probablity), it should be application independent, and it should provide a name-content binding through the use of content hashes as part of the name. The requirement for using DECADE names which are globally unique with a high probability stems from the envisioned usage of hashes. Hashes typically ensure two items will have different hashes with a certain probability, but there is typically a very limited risk that those 2 items will have the same hash value.

A concrete control specification needs to define the concrete name format and possibly also baseline hashing algorithms. The name format MUST be suitable for use in different possible SDT instantiations.

[refs.ni-core] specifies a URI-based name format for naming objects, e.g., through content hashes. NI URIs fundamentally provide an hash algorithm identifier, the actual digest value and can provide additional information such as object type information or locator hints.

ni:///sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc

Figure 2: Example: DECADE NI URI

The NI URI in the example above specifies SHA-256 as the hash

algorithm, and provides the digest of an object. DECADE implementations can generate such names independently, without requiring any infrastructure (when creating objects), and they can verify the name-content binding by calculating the hash of an received object and by comparing the result to the name that was used for the object.

NI URIs can optionally provide authority information, i.e., information about an authority that may assist applications in accessing the object. DECADE should not require authority information to be present.

The NI format allows the optional specification of media types (of the referenced objects) through the addition of parameters:

```
ni:///sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?ct=image/jpeg
```

Figure 3: Example: DECADE NI URI with content type

Such information may be present in URIs, but DECADE should not require such information. It is also important to note that parameters are not considered when testing NI URIs for identity.

NI URIs can be mapped to HTTP URIs, and some HTTP URIs can be mapped to NI URIs. This can be useful for deriving a locator for obtaining NI-named objects without explicit specification. The following example depicts an NI URI with an authority part that is mapped to an HTTP URI (using the well-known convention specified in RFC 5785 [RFC5785]).

```
ni://decade127.example.com/sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6
fkc

http://decade127.example.com/.well-known/ni/sha-256/B_K97zTtFuOhug27fke4_Zg
c4Myz4b_lZNgsQjy6fkc
```

Figure 4: Example: DECADE NI URI mapping to HTTP URI

There are other possibilities to derive the host name part of the HTTP URI (when no autority information is present in the NI URI), e.g., from the application context that the NI URI was used in. For DECADE, we recommend that Application Endpoints that want to refer other Applications to a DECADE object on a specific server (assuming an HTTP-based SDT), provide the server host name as an authority element of the NI URI, as depicted above. It should be noted that this only works with HTTP (or HTTPS)-based SDTs. It is possible to specify additional/alternative locators using the NI parameter mechanisms (which will be described in a future version of this document).

REC_03: There should be exactly one DECADE name format.

REC_04: The DECADE name format must be suitable for use in different possible SDT instantiations.

REC_05: The DECADE names should used the NI URI format.

REC_06: DECADE should allow for different hash algorithms to be used. SHA-256 should be defined as MANDATORY, i.e., all applications that need to validate name-content binding of objects should be able to deal with SHA-256 hash digests.

REC_07: DECADE should allow for different hash algorithms to be used. SHA-256 should be defined as MANDATORY, i.e., all applications that need to validate name-content binding of objects should be able to deal with SHA-256 hash digests.

REC_08: DECADE should allow for different hash algorithms to be used. SHA-256 should be defined as MANDATORY, i.e., all applications that need to validate name-content binding of objects should be able to deal with SHA-256 hash digests.

REC_09: In an application context where SDT==HTTP (or HTTPS), DECADE Application Endpoints should use the authority element in NI URIs to specify a HTTP server name when referring other Application Endpoints to a specific URL.

Figure 5: Recommendations

3.2. Object Addressing

Section Section 3.1 describes how complete objects are potentially named within DECADE. However, it might be also necessary to address parts of a DECADE object, if such objects are accessible in parts. A typical example is the usage of chunks, i.e., equal parts of a file used by peer-to-peer filesharing to exchange data.

This addressing might be required if:

- an object is not completly loaded on a DECADE server, but DECADE clients should be able to retrieve it while the object is being retrieved by the server;
- DECADE clients do only need to access parts of the object, as they have already retrieved some other parts of the object;

- DECADE clients retrieve a particular object from multipe sources at the same time and thus do only require a subset from a particular DECADE server.

REC_10: The DECADE SDT should allow to retrieve parts of an object.
REC_11: The DECADE SDT should allow DECADE servers to specify which parts of an object are available.
REC_12: The DECADE SDT should allow DECADE clients to request single parts or a range of parts from the DECADE server.

Figure 6: Recommendations with respect to Addressing

4. Authentication and Access Control

The DECADE architecture has a concept of token-based authentication and access control. The idea is that Application Endpoints that are referring other Application Endpoints to a DECADE server provide tokens to these other Application Endpoints. Those would use these tokens when communicating with a server, and the tokens would be meaningful to the server for making acess control decisions.

OAuth is one particular candidate mechanism to be used for token-based authentication and access control. (A detailed analysis will be provided in a future version of this document.) A mechanism such as OAuth would be used by HTTP in specific ways, i.e., by using HTTP header fields -- this would be the DRP instantiation for a specific SDT.

Communincating authentication information between Application Endpoints is out of scope for DECADE specifications; it is assumed that this would rely on application-specific protocols. However, there are principally two options:

- authentication tokens in the specific application protocol; or
- authentication tokens in the object identifier that Application Endpoints use to refer other Application Endpoints to a DECADE server.

Including the authentication tokens in the object would provide an application-protocol-independent way for communicating this information between Application Endpoints. The parameter mechanism of NI URIs could be used for that:

ni://decade127.example.com/sha-256;B_K97zTtFuOhug27fke4_Zgc4Myz4b_lZNgsQjy6fkc?auth=AHFK34F

Figure 7: Example: Authentication tokens in NI URIs

For each SDT specification, there needs to be an algorithm to map the authentication token to specific header fields, messages, etc. of the particular protocol.

REC_13: DECADE should use an NI URI parameter for representing authentication information in object identifiers.

Figure 8: Recommendations

5. General SDT Considerations

5.1. Dealing with Application Contexts, Resource Collection and Other Structure

Fundamentally, DECADE is intended to provide access to resources which are distributed in in-network storage servers for different applications. Such resources should be named uniquely across different servers, and the same resource should be accessible at different servers using the same name.

Different servers, different file transfer, and different remote file system protocols may provide different capabilities for organizing resources in hierarchical structures (collections, file system directories etc.). Since DECADE already provides a way to name resources uniquely across different servers and protocols (through the DECADE naming scheme), SDT (and DECADE in general) should not require or rely on any hierarchical name space structure.

Application-specific structure (e.g., collecting all chunks of a specific media resource) should be dealt with on the application layer, i.e., through the use of "torrent files" or index files that reference the DECADE resources.

Similarly, DECADE resources from different application contexts should not be distinguished by additional name components, direcory names etc., since the DECADE naming scheme already provides for a unique naming of resource across application contexts.

Consequently, any operations on remote file system structures, collections etc. should be orthogonal to DECADE and not be supported by SDT. Specific protocols that an SDT instantiation leverages may provide support for that, but we recommend that such operations are considered out of scope for SDT.

5.2. Server-to-Server Communication

The DECADE architecture [refs.decadearch] describes the operation of Server-to-Server Protocols, which are intended to enable DECADE servers to distribute data objects to other servers, without the need of Application Endpoint interaction. One possible way of operation is that an Application Endpoint (client) would upload a data object to a DECADE server, and that server would then upload the object to one or more other servers, thus acting as a client to those other servers. In addition, an Application Endpoint would also be able to request a DECADE server to download the object from another specified server itself.

For specifying a concrete SDT, some design questions need to be taken:

- Is it possible to specify only one or multiple target DECADE servers?
- Most HTTP-based protocols do not support requesting/configuring server-to-server communication natively. We recommend this feature be implemented without changing/extending those protocols.

5.3. Recommendations

REC_14: DECADE should not assume any structure (collections, containers, directories) on DECADE servers.

REC_15: DECADE object identifiers should be flat labels.

REC_16: It should be possible for DECADE server to distribute objects between servers using SDT. An SDT instantiation should provide a corresponding mechanism.

REC_17: DECADE should define a way to specify (control) the distribution of objects between servers.

REC_18: Server-to-Server communication should not require the introduction of new HTTP request types (for HTTP-based SDT).

Figure 9: Recommendations

6. CDMI Considerations

CDMI [refs.CDMI] has been considered as a candidate basis for an DECADE SDT instantiation. This section discusses a few aspects and

potential issues for adopting CDMI.

In general, the assumption is that CDMI (as a certain way to leverage HTTP for accessing and managing cloud data) can be used for DECADE. Since CDMI has many features (namely the data management features) that are not required by DECADE, we assume that a CDMI-based SDT specification would specify a subset of CDMI and specify a list of requirements for implementations on how to use the mechanisms of the subset in detail.

6.1. CDMI Content Type Operations

CDMI provides uploading/downloading/deleting etc. data with CDMI content types and with non-CDMI content types. CDMI content type operations use JSON to encode objects (and meta information), i.e., PUT requests would encode the data object in JSON, and response messages to GET requests would also encode the returned object in JSON. Non-CDMI content type operations may also use JSON for encoding certain information, for example for data object meta information, but the object itself would be transmitted directly in message bodies (as non-CMDI web servers would do).

A CDMI-based SDT should use the non-CDMI content type operations, for efficiency and backwards-compatibility reasons.

6.2. CDMI Features and SDT

CDMI provides a broad range of feature for Cloud Data Management, such as:

- discovering capabilities of a cloud storage provider;
- creating a new container;
- creating a new data object;
- listing the contents of a container;
- reading the contents of a data object;
- reading the value of a data object; and
- deleting a data object.

Moreover, CDMI provides a set of administrative operations, such as:

- managing domain objects representing the concept of administrative ownership (CDMI supports a hierarchy of domains and provides

operations to manage those);

- queue object resource operations, providing first-in, first-out access for storing and retrieving data;
- capability query operations, allowing a client to find out about the subset of CDMI features that a server supports;
- exporting (and configuring the exporting of) data objects to other protocol domains such as NFS, iSCSI, WebDAV etc.;
- serialization and de-serialization of data;
- configure access control through ACLs;
- retention and hold management;
- scope specifications to allow clients to select data objects based on filter/search expressions;
- results specifications (to enable a client to specify subsets of data objects to be returned);
- logging;
- notification queues (for example for notfying clients about changes to a file system or to certain objects); and
- query queues (enabling clients to requests data objects based on meta data or content search expressions).

SDT over CDMI should specify a subset of these features and use the CDMI capability description mechanism to describe the subset of supported features.

6.3. CDMI Containers

Containers are a fundamental concept for CDMI, and they are used for grouping objects. In fact, containers are CDMI objects, and they can be addressed and manipulated using the same CDMI operations that are used for data objects.

With a flat naming scheme (as we expect DECADE to employ) there is no strong need for grouping objects in containers, so we recommend that the containers and container names should not be used for generating DECADE object names.

6.4. Object Identifiers in CDMI

CDMI required globally unique object IDs be used for all objects stored on a CDMI server, which is conceptually similar to the DECADE architecture requirements for naming.

In CDMI, objects are either accessible by their container-based hierarchical named such as "http://decade.example.com/root/vod/video1" or by their object ID such as "http://decade.example.com/root/cdmi_objectid/647284746393", with "647284746393" being the object ID.

CDMI specifies how object IDs should be generated. Object ID are variable length byte sequences with a maximum length of 40 bytes, and they provide the following structure:

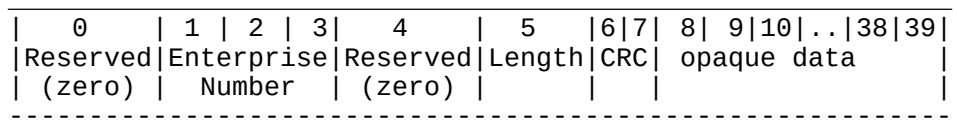


Figure 10: CDMI Object ID structure

Although CDMI Objects IDs could provide content hashes (in the opaque data fields), these IDs are not directly compatible to the current NI URI format. It is possible to convey the additional information of CDMI IDs in NI URIs, employing the extension mechanismsm, but syntactically, the NI URI would be different.

Although applications can treat these IDs as opaque bit strings, the format enables integrity checking for those applications that need it. In CDMI, the assumption is that the *server* generate these IDs, for example upon having received the object from a client over the upload interface. This server-based ID generation is the direct opposite of the client-based ID generation that we expect for DECADE.

6.5. Recommendations for SDT over CDMI

REC_19: The difference between CDMI’s object ID syntax and the NI URI syntax should be addressed by either adapting CDMI’s syntax or by defining a bijective mapping between CDMI and NI URIs.

REC_20: CDMI containers should not be used.

REC_21: CDMI should only by used in the non-CDMI content type mode

Figure 11: Recommendations

7. Security Considerations

Several security considerations need to be investigated for a DECADE SDT protocol and for DECADE in general. First, proper access control to objects stored at DECADE servers must be provided (OAuth is a means to do this, but the specific security implications for using OAuth in the context of DECADE need to be considered, and potential attacks need to be analyzed and described). Second, the potential for Denial-of-Service attacks on DECADE servers must be minimized. Finally, the integrity of data items stored at DECADE servers must be maintained, and clients must have a way to verify the integrity of data items they retrieve from a DECADE server (hash-based or self-certifying schemes as a component of the DECADE name space can be a means to provide these requirements, but the specific implications and potential attacks on data integrity need to be condidered carefully and described in detail). Future versions of this document will study these security aspects in more detail.

Also, SDT over HTTP-based protocols MUST support HTTPS. How applications choose whether to use HTTP or HTTPS will be discussed in a future version of this document.

8. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3744] Clemm, G., Reschke, J., Sedlar, E., and J. Whitehead, "Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol", RFC 3744, May 2004.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.

[refs.CDMI] "CDMI", HTML http://www.snia.org/cdmi, September 2011.

[refs.decadearch] Alimi, R., Yang, Y., Rahman, A., Kutscher, D., and HP.

Liu, "DECADE Architecture", draft-ietf-decade-arch-03 (work in progress), September 2011.

[refs.ni-core]
Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., and P. Hallam-Baker, "The Named Information (ni) URI Scheme: Core Syntax", draft-farrell-decade-ni-00 (work in progress), October 2011.

Appendix A. Acknowledgments

Dirk Kutscher is partially supported by the SAIL project (Scalable and Adaptive Internet Solutions, http://www.sail-project.eu), a research project supported by the European Commission under its 7th Framework Program (contract no. 257448).

Jan Seedorf and Martin Stiemerling are partially supported by the COAST project (COntent Aware Searching, retrieval and sTreaming, http://www.coast-fp7.eu), a research project supported by the European Commission under its 7th Framework Program (contract no. 248036).

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the SAIL or the COAST project or the European Commission.

Authors' Addresses

Dirk Kutscher
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: kutscher@neclab.eu
URI: http://dirk-kutscher.info/

Martin Stiemerling
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: martin.stiemerling@neclab.eu
URI: http://ietf.stiemerling.org

Jan Seedorf
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: seedorf@neclab.eu

DECADE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

D. Wang
Huawei Technologies
Y. Yang
Yale University
Oct 31, 2011

An HTTP-based Decade Resource Protocol
draft-wang-decade-drp-01

Abstract

This document explorers the design of an HTTP-based DECADE Resource Protocol (DRP), which can be used for content and resource management between a DECADE Client and a DECADE Server, or between two DECADE Servers. We identify the function entities, and present initial protocol message flow and syntax design. The purpose of this document is to get design discussion started, not to provide a complete solution.

Table of Contents

1. Introduction 3
2. Terminologies and concepts 3
3. Protocol Overview 3
3.1. Overview 3
3.2. Function Entities 4
3.3. Protocol Architecture 4
3.4. Object Naming 4
3.5. Protocol Operations 4
4. Message Syntax and Processing 5
4.1. Encoding 5
4.2. Common Message Processing 6
4.3. DECADE Messages 6
4.3.1. Transport_Query Message 6
4.3.2. Access_Token Message 7
4.3.3. Server_States_Query Message 8
4.3.4. Object_Property_Query Message 10
4.3.5. Object_Property_Set Message 12
4.3.6. Delete_Data Message 13
4.4. Error Response Messages 14
5. Integration with an HTTP-based SDT 15
5.1. Put_Object Message 15
5.2. Get_Object Message 16
6. Security Considerations 18
7. IANA Considerations 18
8. Acknowledgements 18
9. References 19
9.1. Normative Reference 19
9.2. Informative Reference 19
Authors' Addresses 19

1. Introduction

The DECADE Architecture document [I-D.ietf-decade-arch-03] identifies a DECADE architecture that consists of two logically independent protocols: the DECADE Resource Protocol (DRP) and the Standard Data Transport (SDT). The formal provides access control and resource scheduling between two DECADE peers, while the latter is used to transfer data objects to and from a DECADE Server. The architecture document observes that the two protocols may be realized on the same wire.

In this document, we present an initial design of a DRP based on HTTP. We use the DECADE requirements [I-D.ietf-decade-reqs-04] to guide our design. The purpose of this document is to get design discussions started, not to provide a complete solution.

Specifically, the DRP we present provides a signaling layer for transport negotiation, content management, access control and resource control. The SDT is responsible for data put, retrieve, delete, and metadata update, and the implementation of access control and resource control policies.

We also show that the HTTP based DRP may be extended to include an HTTP based SDT, achieving a same-wire design, although it is also possible to use another SDT.

2. Terminologies and concepts

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "MAY", "MAY NOT" and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This draft uses the terms defined in [I-D.ietf-decade-reqs-04] and in [I-D.ietf-decade-arch-03].

DECADE peers: Entities involved in the DRP, such as a DECADE Server or a DECADE Client.

3. Protocol Overview

3.1. Overview

DECADE Protocol is spoken between DECADE Client and DECADE Server or between DECADE Servers to manage, store, and retrieve data objects for data distribution. A DECADE Client can be an application server, or embedded in a web browser, etc.

DECADE needs a mandatory naming scheme, that can be used to achieve content security and de-duplication. In this initial design, we assume that the name space is flat for simplicity. We anticipate that the name scheme should also allow for group operations on a set of data objects with a similar/same attribute.

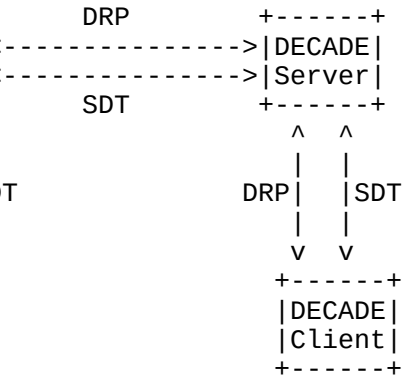
3.2. Function Entities

We consider two primary function entities involved in DECADE: the DECADE Server and the DECADE Client.

DECADE Server: A DECADE Server stores DECADE data inside the network, and thereafter manages both the stored data and access to that data [I-D.ietf-decade-arch-02].

DECADE Client: A DECADE Client stores and retrieves data at DECADE Servers [I-D.ietf-decade-arch-02].

3.3. Protocol Architecture



3.4. Object Naming

(To be added...)

3.5. Protocol Operations

The DRP we present is a request-response protocol. Requests and corresponding responses are exchanged between DECADE Client and Server or between DECADE Servers.

In particular, we identify the following operations:

Access Token: This operation allows the DECADE Server to limit access and resource control of data objects stored on it via authenticated tokens. The token should limit permitted operations to permitted objects by permitted clients during permitted period, as well as priority for bandwidth given to requested operation, and so on. OAUTH is adopted here to realize access and resource control.

Query Server Statue (from the system's view): This operation allows a DECADE Client to query statue information of a specific DECADE Server from the server view, e.g., lists of associated objects, resource used/available, and so on.

Query Server Statue (from the user's view): This operation allows a DECADE Client to query its related statue information of a specific DECADE Server, e.g., list of associated objects the DECADE Client stored in the Server, the Client's available resources, and so on.

Query Object Property: This operation allows a DECADE Client to query object properties with certain authentication, including TTL of object, object size, or object type.

Set Object Property: This operation allows a DECADE Client to set data object properties with certain authentication.

Delete Data Object: This operation is used to delete data objects from a DECADE Server.

Complementing the preceding DRP operations, the following SDT operations may be defined for a complete DECADE Protocol:

Put Data Object: This operation is used to write data objects to a DECADE Server.

Get Data Object: This operation is used to download data objects from a DECADE Server.

4. Message Syntax and Processing

We now present the encoding and processing.

4.1. Encoding

The DRP in this document follows the standard request and response formats for HTTP Request and Response messages [RFC2616].

Specifically, the header fields in both request and response messages follows the standard rules for HTTP/1.1 Header fields, which MAY be included in the messages if necessary.

We use JSON to encode the bodies for both request and response messages.

In the following messages, both *** and XXX represent data to be insert, either numeric data or alphanumeric data.

4.2. Common Message Processing

After receiving a DECADE protocol message, some basic processing will be performed, regardless of the message type and the receiving entity.

Upon receiving a message, the message is examined to ensure that it is properly formed, which is done by the receiver itself. If not, appropriate standard HTTP errors MUST be generated.

If the message is found to be incorrectly formed or the length does not match the length encoded in the header, the receiver MUST reply with an HTTP 400 response.

If the version number is not supported by the receiver, the receiver MUST reply with an HTTP 400 response.

If the receiver is unable to process the message temporarily because it is in an overloaded state, the receiver SHOULD reply with an HTTP 503 response.

If the receiver encounters an internal error while attempting to process the message, the receiver MUST generate an HTTP 500 response.

4.3. DECADE Messages

4.3.1. Transport_Query Message

After the DECADE Client locating its authenticated DECADE Server, transport capability query MUST be performed between them. A Transport_Query message is used to query the DECADE server about supported transports. A DECADE client will send a request to the DECADE server to ask for the transport protocols supported by the server. And then the DECADE server will send back a reply with the supported transport protocols. DECADE server must listen on the server ports of the supported protocols (e.g. NFS). By that, the DECADE client (active) can impose data operations on the DECADE server using either one of the supported transport protocols

supported by the DECADE server. Below is an example Transport_Query message:

```
GET /decade/transport/ HTTP/1.1
Host:  example.com
Content-Length: ***
Content-Type: application/decade-transport-req+json
X-DECADE-Protocol-Version: 1.0
```

If the query message is valid, the DECADE Server sends back a response listing the supported SDT protocols. An example is shown below:

```
HTTP/1.1 200 OK
Content-Length: ***
Content-Type: application/decade-transport-ans+json
X-DECADE-Protocol-Version: 1.0

{
"transport-protocol":
        {
                "HTTP": true;
                "NFS": true;
                "WebDAV": true;
        }
}

JSON Object:
Object {
        JSONBool HTTP;     [OPTIONAL]
        JSONBool NFS;      [OPTIONAL]
        JSONBool WebDav;   [OPTIONAL]
} TransportProtocol;

Object {
    TransportProtocol transport-protocol;
}
```

When the DECADE Client receive the response from the Server, it will choose a certain protocol it can support and then communicate with the Server via the chosen protocol.

4.3.2. Access_Token Message

This operation allows a DECADE Client to obtain an access token from a DECADE Server to limit access and achieve resource control.

Specifically, an access token should limit permitted operations to permitted operations to permitted objects by permitted clients during permitted period, as well as priority for bandwidth given to

requested operation, and so on.

In DECADE architecture, a user delegation scheme is proposed. We anticipate that a DECADE Client can generate such Access Token locally as well.

An access token to be generated may be depended on the SDT specified, should the token to be carried by the SDT operation.

(TBA)

4.3.3. Server_States_Query Message

This operation allows state query. The state can be either system-wide or on a particular user. An example message is:

```
POST /decade/state HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-stat-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: XXX

{
        "state-request":
        {
                "StorageUsed": true;
                "StorageAvailable": true;
                "BandwidthUsed": true;
                "BandwidthAvailable": true;
                "ObjectNum": true;
                "ObjectList": true;
        }

        "user-type":
        {
                "System": true;
        }

        "token":XXX;
}
```

```
JSON Object:
Object{
        JSONBool StorageUsed;           [OPTIONAL]
        JSONBool StorageAvaliable;      [OPTIONAL]
        JSONBool BandwidthUsed;         [OPTIONAL]
        JSONBool BandwidthAvailable;    [OPTIONAL]
        JSONBool ObjectNum;             [OPTIONAL]
```

```
        JSONBool ObjectList;            [OPTIONAL]
   }ServerState;

   Object{
        JSONBool System;        [OPTIONAL]
        JSONBool User;          [OPTIONAL]
   }UserType;

   Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
   }AuthToken;

   Object{
        ServerState server-state;
   }

   Object{
        UserType user-type;
   }

   Object{
        AuthToken token;
   }
```

If a state query message is valid, the DECADE Server responses with corresponding server state. If the "user-type" in the message body is "System", then the server should return the status of resource consumption on this server after verifying the user's access right. Usually the right will be authorized to the system administrator. If the "user-type" in the message body is "User", then the server should return the status of resource consumption of this particular user on this server after verifying the user's access right. An example of the Response is as following:

```
HTTP/1.1 200 OK
Content-Length: ***
Content-Type: application/decade-stat-ans+json
X-DECADE-Protocol-Version: 1.0

{
        "server-state":
        {
                "storage-used": ***;
                "storage-available": ***;
                "bandwidth-used": ***;
                "bandwidth-available": ***;
                "object-number": ***;

                "object-list":
                [
                        "data-object":
                        {
                                "object-name":XXX;
                        }
                ]
        }
}

JSON Object:
Object{
        JSONNumber storage-used;                [OPTIONAL]
        JSONNumber storage-avaliable;           [OPTIONAL]
        JSONNumber bandwidth-used;              [OPTIONAL]
        JSONNumber object-number;               [OPTIONAL]
        JSONArray object-list["DataObject"];    [OPTIONAL]
}ServerState;

Object{
        JSONString object-name;
}DataObject;

Object{
        DataObject data-object;
}
```

4.3.4. Object_Property_Query Message

This operation allows a DECADE Client to query object properties with certain authentication, including TTL of object, object size, and object type. The request message is as following:

```
POST /decade/object/<obj-name>/property HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-objprop-query-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: XXX

{
        "token": XXX;
}

JSON Object:
Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        AuthToken token;
}
```

If the object property query message is valid and successfully proceeded, the DECADE Server replies the request info. An example of the response message is as below:

```
HTTP/1.1 200 OK
Content-Length: ***
Content-Type: application/decade-objprop-query-ans+json
X-DECADE-Protocol-Version: 1.0

{
        "object-property":
        {
                "object-type": XXX;
                "object-size": ***;
                "owner": XXX;
                "TTL": ***;
        }
}

JSON Object:
Object{
        JSONString object-type; [OPTIONAL]
        JSONNumber object-size; [OPTIONAL]
        JSONString owner;       [OPTIONAL]
        JSONNumber TTL;         [OPTIONAL]
}ObjectPerperty;

Object{
        ObjectProperty object-property;
}
```

4.3.5. Object_Property_Set Message

This operation allows a DECADE Client to set object properties (metadata). The request message is as following:

```
PUT /decade/<obj-name>/property HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-objprop-set-req+json
X-DECADE-Protocol-Version: 1.0
X-DEADE-TOKEN: XXX

{
        "object-property":
        {
                "TTL": ***;
        }

        "token": XXX;
}

JSON Object:
Object{
        JSONNumber TTL;         [OPTIONAL]
}ObjectProperty;

Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        ObjectProperty object-property;
}

Object{
        AuthToken token;
}
```

If the object property set message is valid, the DECADE Server replies with the following message:

```
HTTP/1.1 204 NO CONTENT
Content-Length: ***
Content-Type: application/decade-objprop-set-req-ans+json
X-DECADE-Protocol-Version: 1.0
```

4.3.6. Delete_Data Message

This operation is used to delete objects from a DECADE Server. An example of the request message is as following:

```
DELETE /decade/<obj-name> HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-deledata-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: XXX

{
        "token": XXX;
}

Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        AuthToken token;
}
```

An example of the response message is as following:

```
HTTP/1.1 204 NO CONTENT
Content-Length: ***
Content-Type: application/decade-deledata-req+json
X-DECADE-Protocol-Version: 1.0
```

4.4. Error Response Messages

The error response messages for the DECADE protocol are described below:

SUCCESSFUL (200 OK): a message has been processed properly and the desired operation has completed. If the message is a request for information, the body will also include the requested information.

NO CONTENT (204 NO CONTENT): The server successfully processed the request, but is not returning any content.

INVALID SYNTAX (400 Bad Request): Indicates an error in the format of the message/message body.

VERSION NOT SUPPORTED (400 Bad Request): Invalid version of the protocol or message bodies.

AUTHENTICATION REQUIRED (401 UNAUTHORISED): Authentication is

required to access this information.

MESSAGE FORBIDDEN (403 FORBIDDEN): The requester is not allowed to make this request.

OBJECT NOT FOUND (404 NOT FOUND): The requested object cannot be found.

SERVER NOT FOUND (404 NOT FOUND): The requested server cannot be found.

5. Integration with an HTTP-based SDT

5.1. Put_Object Message

This operation is used to write objects to a DECADE Server. An example of the request message is as following:

```
PUT /decade/<obj-name> HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-putdata-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: XXX

{
        "object-list":
        [
                "data-object":
                {
                        "object-size": ***;
                        "object-type": XXX;

                        "object-metadata":
                        {
                                "object-type": XXX;
                                "object-size": ***;
                                "TTL": ***;
                        }
                }
        ]

        "token": XXX;
}
```

JSON Object:

```
Object{
```

```
            JSONNumber object-size;         [OPTIONAL]
            JSONSting object-type;          [OPTIONAL]
            JSONObject object-metadata:     [OPTIONAL]
            {
                    JSONString owner;
                    JSONNumber TTL;         [OPTIONAL]
            }
    }DataObject;


    Object{
            JSONArray object-list["DataObject"];
    }

    Object{
            ObjectOperation permitted-operations<0..*>;
            JSONObject permitted-objects;
            JSONObject permitted-clients;
            JSONNumber expiration-time;
            JSONNumber priority;
    }AuthToken;

    Object{
            AuthToken token;
    }
```

An example of response message is as below:

```
HTTP/1.1 204 NO CONTENT
Content-Length: ***
Content-Type: application/decade-putdata-ans+json
X-DECADE-Protocol-Version: 1.0
```

5.2. Get_Object Message

This operation is used to download objects from a DECADE Server. An example of the request message is as following:

```
POST /decade/<obj-name> HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-getdata-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: XXX

{
        "token": XXX;
}

Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        AuthToken token;
}
```

An example of the response message is as following:

```
HTTP/1.1 200 OK
Content-Type: application/decade-getdata-ans+json
X-DECADE-Protocol-Version: 1.0

{
        "data-object":
        {
                "object-size": ***;
                "object-type": XXX;
                "object-metadata":
                {
                        "owner": XXX;
                        "TTL": ***;
                }
        }
}

JSON Object:
Object{
        JSONNumber object-size;         [OPTIONAL]
        JSONSting object-type;          [OPTIONAL]
        JSONObject object-metadata:     [OPTIONAL]
        {
                JSONString owner;
                JSONNumber TTL;         [OPTIONAL]
        }
}DataObject;

Object{
        DataObject data-object;
}
```

6. Security Considerations

The security considerations described in both [I-D.ietf-decade-arch] and [I-D.ieft-decade-problem-statement] apply to this draft as well.

7. IANA Considerations

This draft does not have any IANA considerations.

8. Acknowledgements

We would like to express our sincere thanks and appreciation to Zhou Hong for his valuable suggestions and support.

9. References

9.1. Normative Reference

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", June 1999.

9.2. Informative Reference

[I-D.ietf-decade-reqs] Gu, Y., Bryan, D., Yang, Y., and R. Alimi, "DECADE Requirements", September 2011.

[I-D.ietf-decade-arch] Alimi , R., Yang, Y., Rahman, A., Kutscher, D., and H. Liu, "DECADE Architecture", July 2011.

[I-D.ietf-decade-problem-statement] Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement".

Authors' Addresses

Wang Danhua
Huawei Technologies
No. 101 Software Avenue
Nanjing, Jiangsu Province 210001
P.R.China

Phone: +86-25-56624734
Fax: +86-25-56624702
Email: wangdanhua@huawei.com

Y.Richard Yang
Yale University

Email: yry@cs.yale.edu

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 13, 2011

D. Wang
Huawei Technologies
H. Liu
Y. Yang
Yale University
July 16, 2012

An HTTP-based Decade Resource Protocol
draft-wang-decade-drp-04

Abstract

This document explorers the design of an HTTP-based DECADE Resource Protocol (DRP), which can be used for content and resource management between a DECADE Client and a DECADE Server, or between two DECADE Servers. We identify the function entities, and present initial protocol message flow and syntax design. The purpose of this document is to get design discussion started, not to provide a complete solution.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 13, 2011.

Table of Contents

1. Introduction 3
2. Terminologies and concepts 3
3. Protocol Overview 3
3.1. Overview 3
3.2. Function Entities 4
3.3. Protocol Architecture 4
3.4. Object Naming 4
3.5. Protocol Operations 4
4. Access and Resource Control 5
5. Token Structure/HTTP Authentication Format 6
6. Message Syntax and Processing 7
6.1. Encoding 7
6.2. Common Message Processing 7
6.3. DECADE Messages 8
6.3.1. Transport_Query Message 8
6.3.2. Access_Token Message 9
6.3.3. Server_States_Query Message 10
6.3.4. Object_Property_Query Message 12
6.3.5. Object_Property_Set Message 14
6.3.6. Delete_Data Message 15
6.4. Error Response Messages 16
7. Integration with an HTTP-based SDT 17
7.1. Put_Object Message 17
7.2. Get_Object Message 18
8. Remote_Get_Object Message 19
9. Security Considerations 20
10. IANA Considerations 20
11. Acknowledgements 20
12. References 21
12.1. Normative Reference 21
12.2. Informative Reference 21
Authors' Addresses 21

1. Introduction

The DECADE Architecture document [I-D.ietf-decade-arch-03] identifies a DECADE architecture that consists of two logically independent protocols: the DECADE Resource Protocol (DRP) and the Standard Data Transport (SDT). The former provides access control and resource scheduling between two DECADE peers, while the latter is used to transfer data objects to and from a DECADE Server. The architecture document observes that the two protocols may be realized on the same wire.

In this document, we present an initial design of a DRP based on HTTP. We use the DECADE requirements [I-D.ietf-decade-reqs-04] to guide our design. The purpose of this document is to get design discussions started, not to provide a complete solution.

Specifically, the DRP we present provides a signaling layer for transport negotiation, content management, access control and resource control. The SDT is responsible for data put, retrieve, delete, and metadata update, and the implementation of access control and resource control policies.

We also show that the HTTP based DRP may be extended to include an HTTP based SDT, achieving a same-wire design, although it is also possible to use another SDT.

2. Terminologies and concepts

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "MAY", "MAY NOT" and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This draft uses the terms defined in [I-D.ietf-decade-reqs-04] and in [I-D.ietf-decade-arch-03].

DECADE peers: Entities involved in the DRP, such as a DECADE Server or a DECADE Client.

3. Protocol Overview

3.1. Overview

The DECADE Protocol is spoken between a DECADE Client and a DECADE Server or between two DECADE Servers to manage, store, and retrieve data objects for data distribution. A DECADE Client can be an application server, or embedded in a web browser, etc.

We use the scheme where DECADE uses a data-derived naming scheme to achieve content security and de-duplication. In this initial design, we assume that the name space is flat for simplicity. We anticipate that the naming scheme may allow for group operations on a set of data objects with a similar/same attribute.

3.2. Function Entities

We consider two primary function entities involved in DECADE: the DECADE Server and the DECADE Client.

DECADE Server: A DECADE Server stores DECADE data inside the network, and thereafter manages both the stored data and access to that data [I-D.ietf-decade-arch-02].

DECADE Client: A DECADE Client stores and retrieves data at DECADE Servers [I-D.ietf-decade-arch-02].

3.3. Protocol Architecture

Figure 1 presents the simple protocol architecture.

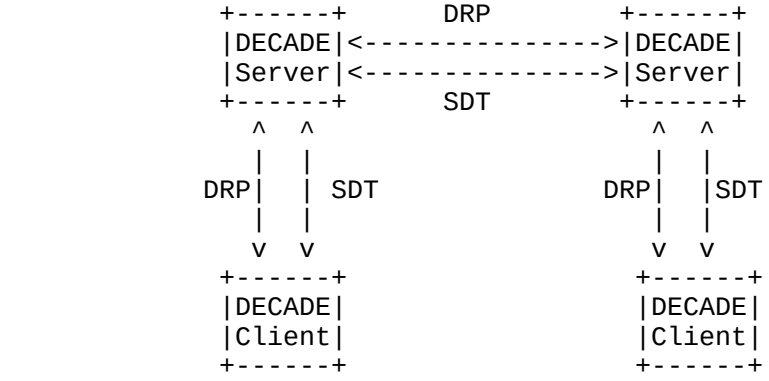


Figure1 Protocol Architecture

3.4. Object Naming

(To be added...)

3.5. Protocol Operations

The DRP we present is a request-response protocol. Requests and corresponding responses are exchanged between a DECADE Client and a DECADE Server or between two DECADE Servers. A request includes an Access Token, which allows the DECADE Server to limit access and implement resource control. In particular, a token specifies permitted operations to permitted objects by permitted clients during

permitted period, as well as priority for bandwidth given to requested operation, and so on.

In particular, we identify the following management requests:

Query Server Status (from the system's view): This operation allows a DECADE Client to query statue information of a specific DECADE Server from the server view, e.g., lists of associated objects, resource used/available, and so on.

Query Server Status (from the user's view): This operation allows a DECADE Client to query status specific to itself, e.g., list of associated objects that the DECADE Client stored in the Server, the Client's available resources, and so on.

Query Object Property: This operation allows a DECADE Client to query object properties, including TTL of objects, object sizes.

Set Object Property: This operation allows a DECADE Client to set data object properties.

Delete Data Object: This operation is used to delete data objects from a DECADE Server.

Complementing the preceding DRP operations, we define the following SDT operations archieve a complete DECADE Protocol:

Put Data Object: This operation is used to write data objects to a DECADE Server.

Get Data Object: This operation is used to download data objects from a DECADE Server.

4. Access and Resource Control

We start with a brief evaluation on the feasibility of using an existing IETF protocol for resource control in DECADE. In particular, we evaluate the feasibility of extending an existing protocol for granting a third party access to the resource and data objects owned by a resource/data owner. We consider the applicability of OAuth, AAA, and Kerberos.

In Kerberos [RFC4120], a client obtains Tickets to obtain services. Specifically, in Kerberos, a client first obtains a Client/TGS session key and a Ticket-Granting-Ticket from an authentication server. An issue of this approach, however, is that it requires a client to authenticate with the system. Hence, it may work well in a

single DECADE domain system where each client has an account at a DECADE provider. On the other hand, DECADE should also work in an inter-domain setting, where a client may not have an account on a Wang, et al. Expires September 13, 2012 [Page 5] Internet-Draft DRP Mar 2012 DECADE server. Hence, extensions could be challenging.

Radius [RFC 2865] [RFC 2866] and its successor Diameter [RFC 3588] are the base AAA protocols. Although extending the binary attribute-value-pairs (AVPs) may be possible to grant network resources for data access, the resource control communication points in the DECADE environment are application clients/servers and DECADE servers, and hence a text-based protocol may be preferred by the application clients. However, this should not prevent that the Diameter protocol be used between a DECADE server and an application server for AAA purposes.

OAuth v2 [draft-ietf-oauth-protocol-v2-23] is used to grant access to the resource owner's resources from a third party without explicitly exposing the resource owner's credentials. It currently focuses on the context of HTTP [RFC2616]. The three-leg-style access assumption is different from the access requirement in the DECADE system environment. Howerver, some grant types may be adopted (such as implicit grant) and extensions can be added for resource control.

5. Token Structure/HTTP Authentication Format

A primary use case for DECADE is that a DECADE Client authorizes other DECADE Clients to store or retrieve data objects from its DECADE storage. Therefore, we explore a DECADE protocol using a token-based authorization scheme instead of explicitly manipulating an Access Control List (ACL) for each DECADE data object.

Tokens can be generated by a DECADE Client itself or any of its trusted parties/entities. After a token has been generated, it can be distributed to other DECADE Clients for them to request for access to the DECADE Server. In this way, token can support anonymous access, and a DECADE Server does not need to know the identity of each DECADE Client that accesses it.

In order to provide a complete authorization scheme, the token MUST carry the following information: Permitted operations, Permitted objects, Permitted Clients, Expiration time, Priority for bandwidth given to requested operation, and Amount of data that may be read or written.

The token can be carried with the request message. A possibility of a header for signature is:

X-DECADE-TOKEN: DECADE keyID;signature

The "signature" represents alphanumeric or numeric data to be inserted, which includes necessary authentication information described before. A design is that the siganture is calculated based on HMAC-SHA1.

signature = Base64(HMAC-SHA1(UTF-8-Encoding-Of(StringToSign)))

StringToSign = PermittedOperation1 + "\n" + ... + PermittedOperationm + "\n" + PermittedObject1 + "\n" + ... + PermittedObjectn + "\n" + TTL + "\n" + Priority for bandwidth given to requested operation + "\n" + Amount of data that may be read or written + "\n"key

After the DECADE Server receives a request, it verifies the authorization by checking the token. If the token is valid, the request/certain operations will be authenticated, otherwise, it will be rejected. If the token is successfully validated, the DECADE Server applies the resources control policies indicated in the token while performing the operation.

6. Message Syntax and Processing

We now present the encoding and processing.

6.1. Encoding

The DRP in this document follows the standard request and response formats for HTTP Request and Response messages [RFC2616].

Specifically, the header fields in both request and response messages follows the standard rules for HTTP/1.1 Header fields, which MAY be included in the messages if necessary.

We use JSON to encode the bodies for both request and response messages.

In the following messages, both *** and XXX represent data to be insert, either numeric data or alphanumeric data.

6.2. Common Message Processing

After receiving a DECADE protocol message, a DECADE server performs some basic processing, regardless of the message type and the receiving entity.

Specifically, upon receiving a message, the DECADE server ensures

that the message is properly formed. If not, appropriate standard HTTP errors MUST be generated. Below are some examples.

If the message is found to be incorrectly formed or the length does not match the length encoded in the header, the receiver MUST reply with an HTTP 400 response.

If the version number is not supported by the receiver, the receiver MUST reply with an HTTP 400 response.

If the receiver is unable to process the message temporarily because it is in an overloaded state, the receiver SHOULD reply with an HTTP 503 response.

If the receiver encounters an internal error while attempting to process the message, the receiver MUST generate an HTTP 500 response.

6.3. DECADE Messages

DECADE protocol can be divided into two layers, the signaling layer and the transport layer. The signaling layer is responsible for transport negotiation, data management, access control and resource control. The Transport layer is responsible for data put, retrieve, delete, metadata update, and the implementation of access control and resource control policies. Therefore, SDT messages must be extended to include tokens.

6.3.1. Transport_Query Message

A Transport_Query message is used to query a DECADE server about its supported transports. A DECADE client will send a request to the DECADE server to ask for the transport protocols supported by the server. And then the DECADE server will send back a reply with the supported transport protocols. A DECADE server must listen on the server ports of the supported protocols (e.g. NFS). By that, the DECADE client (active) can impose data operations on the DECADE server using either one of the supported transport protocols supported by the DECADE server. Below is an example Transport_Query message:

```
GET /decade/transport/ HTTP/1.1
Host:  example.com
Content-Length: ***
Content-Type: application/decade-transport-req+json
X-DECADE-Protocol-Version: 1.0
```

If the query message is valid, the DECADE Server sends back a response listing the supported SDT protocols. An example is shown below:

```
HTTP/1.1 200 OK
Content-Length: ***
Content-Type: application/decade-transport-ans+json
X-DECADE-Protocol-Version: 1.0

{
"transport-protocol":
        {
                "HTTP": true;
                "NFS": true;
                "WebDAV": true;
        }
}

JSON Object:
Object {
        JSONBool HTTP;     [OPTIONAL]
        JSONBool NFS;      [OPTIONAL]
        JSONBool WebDav;   [OPTIONAL]
} TransportProtocol;

Object {
    TransportProtocol transport-protocol;
}
```

When the DECADE Client receives the response from the Server, it will choose a protocol that it supports and communicates with the Server via the chosen protocol.

6.3.2. Access_Token Message

It is possible for a client to ask a DECADE server to generate a key, instead of generating locally. An example message is as following:

```
POST /decade/token HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-access-token-req+json
X-DECADE-Protocol-Version: 1.0

{
    JSONString: DECADEClientId;
    token parameters
}
```

If the DECADE Client identifier is valid and the request message is successfully proceeded, the DECADE Server will reply with the authenticate token. An example of the response message is as below:

```
HTTP/1.1 200 OK
Content-Type: application/decade-access-token-ans+json
X-DECADE-Protocol-Version: 1.0

{
    JSONString: token;
}
```

6.3.3. Server_States_Query Message

This operation allows state query. The state can be either system-wide or on a particular user. An example message is:

```
POST /decade/state HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-stat-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: DECADE signature

{
        "state-request":
        {
                "StorageUsed": true;
                "StorageAvailable": true;
                "BandwidthUsed": true;
                "BandwidthAvailable": true;
                "ObjectNum": true;
                "ObjectList": true;
        }

        "user-type":
        {
                "System": true;
        }

        "token":XXX;
}

JSON Object:
Object{
        JSONBool StorageUsed;           [OPTIONAL]
        JSONBool StorageAvailable;      [OPTIONAL]
        JSONBool BandwidthUsed;         [OPTIONAL]
        JSONBool BandwidthAvailable;    [OPTIONAL]
        JSONBool ObjectNum;             [OPTIONAL]
        JSONBool ObjectList;            [OPTIONAL]
}ServerState;
```

```
Object{
        JSONBool System;        [OPTIONAL]
        JSONBool User;          [OPTIONAL]
}UserType;

Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        ServerState server-state;
}

Object{
        UserType user-type;
}

Object{
        AuthToken token;
}
```

If a state query message is valid, the DECADE Server responses with corresponding server state. If the "user-type" in the message body is "System", then the server should return the status of resource consumption on this server after verifying the user's access right. Usually the right will be authorized to the system administrator. If the "user-type" in the message body is "User", then the server should return the status of resource consumption of this particular user on this server after verifying the user's access right. An example of the Response is as following:

```
HTTP/1.1 200 OK
Content-Length: ***
Content-Type: application/decade-stat-ans+json
X-DECADE-Protocol-Version: 1.0

{
        "server-state":
        {
                "storage-used": ***;
                "storage-available": ***;
                "bandwidth-used": ***;
                "bandwidth-available": ***;
                "object-number": ***;

                "object-list":
                [
                        "data-object":
                        {
                                "object-name":XXX;
                        }
                ]
        }
}

JSON Object:
Object{
        JSONNumber storage-used;                [OPTIONAL]
        JSONNumber storage-available;           [OPTIONAL]
        JSONNumber bandwidth-used;              [OPTIONAL]
        JSONNumber object-number;               [OPTIONAL]
        JSONArray object-list["DataObject"];    [OPTIONAL]
}ServerState;

Object{
        JSONString object-name;
}DataObject;

Object{
        DataObject data-object;
}
```

6.3.4. Object_Property_Query Message

This operation allows a DECADE Client to query object properties with certain authentication, including TTL of object, object size, and object type. The request message is as following:

```
POST /decade/object/<obj-name>/property HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-objprop-query-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: DECADE signature

{
        "token": XXX;
}

JSON Object:
Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        AuthToken token;
}
```

If the object property query message is valid and successfully proceeded, the DECADE Server replies the request info. An example of the response message is as below:

```
HTTP/1.1 200 OK
Content-Length: ***
Content-Type: application/decade-objprop-query-ans+json
X-DECADE-Protocol-Version: 1.0

{
        "object-property":
        {
                "object-type": XXX;
                "object-size": ***;
                "owner": XXX;
                "TTL": ***;
        }
}

JSON Object:
Object{
        JSONString object-type; [OPTIONAL]
        JSONNumber object-size; [OPTIONAL]
        JSONString owner;       [OPTIONAL]
        JSONNumber TTL;         [OPTIONAL]
}ObjectProperty;

Object{
        ObjectProperty object-property;
}
```

6.3.5. Object_Property_Set Message

This operation allows a DECADE Client to set object properties (metadata). The request message is as following:

```
PUT /decade/<obj-name>/property HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-objprop-set-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: DECADE signature

{
        "object-property":
        {
                "TTL": ***;
        }

        "token": XXX;
}

JSON Object:
Object{
        JSONNumber TTL;         [OPTIONAL]
}ObjectProperty;

Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        ObjectProperty object-property;
}

Object{
        AuthToken token;
}
```

If the object property set message is valid, the DECADE Server replies with the following message:

```
HTTP/1.1 204 NO CONTENT
Content-Length: ***
Content-Type: application/decade-objprop-set-req-ans+json
X-DECADE-Protocol-Version: 1.0
```

6.3.6. Delete_Data Message

This operation is used to delete objects from a DECADE Server. An example of the request message is as following:

```
DELETE /decade/<obj-name> HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-deledata-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: DECADE signature

{
        "token": XXX;
}

Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        AuthToken token;
}
```

An example of the response message is as following:

```
HTTP/1.1 204 NO CONTENT
Content-Length: ***
Content-Type: application/decade-deledata-req+json
X-DECADE-Protocol-Version: 1.0
```

6.4. Error Response Messages

The error response messages for the DECADE protocol are described below:

SUCCESSFUL (200 OK): a message has been processed properly and the desired operation has completed. If the message is a request for information, the body will also include the requested information.

NO CONTENT (204 NO CONTENT): The server successfully processed the request, but is not returning any content.

INVALID SYNTAX (400 Bad Request): Indicates an error in the format of the message/message body.

VERSION NOT SUPPORTED (400 Bad Request): Invalid version of the protocol or message bodies.

AUTHENTICATION REQUIRED (401 UNAUTHORIZED): Authentication is

required to access this information.

MESSAGE FORBIDDEN (403 FORBIDDEN): The requester is not allowed to make this request.

OBJECT NOT FOUND (404 NOT FOUND): The requested object cannot be found.

SERVER NOT FOUND (404 NOT FOUND): The requested server cannot be found.

7. Integration with an HTTP-based SDT

7.1. Put_Object Message

This operation is used to write objects to a DECADE Server. An example of the request message is as following:

```
PUT /decade/<obj-name> HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-putdata-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: DECADE signature

{
        "object-list":
        [
                "data-object":
                {
                        "object-size": ***;
                        "object-type": XXX;

                        "object-metadata":
                        {
                                "object-type": XXX;
                                "object-size": ***;
                                "TTL": ***;
                        }
                }
        ]

        "token": XXX;
}
```

JSON Object:

Object{

```
            JSONNumber object-size;         [OPTIONAL]
            JSONString object-type;          [OPTIONAL]
            JSONObject object-metadata:     [OPTIONAL]
            {
                    JSONString owner;
                    JSONNumber TTL;         [OPTIONAL]
            }
    }DataObject;

    Object{
            JSONArray object-list["DataObject"];
    }

    Object{
            ObjectOperation permitted-operations<0..*>;
            JSONObject permitted-objects;
            JSONObject permitted-clients;
            JSONNumber expiration-time;
            JSONNumber priority;
    }AuthToken;

    Object{
            AuthToken token;
    }
```

An example of response message is as below:

```
HTTP/1.1 204 NO CONTENT
Content-Length: ***
Content-Type: application/decade-putdata-ans+json
X-DECADE-Protocol-Version: 1.0
```

7.2. Get_Object Message

This operation is used to download objects from a DECADE Server. An example of the request message is as following:

```
POST /decade/<obj-name> HTTP/1.1
Host: example.com
Content-Length: ***
Content-Type: application/decade-getdata-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: DECADE signature

{
        "token": XXX;
}

Object{
        ObjectOperation permitted-operations<0..*>;
        JSONObject permitted-objects;
        JSONObject permitted-clients;
        JSONNumber expiration-time;
        JSONNumber priority;
}AuthToken;

Object{
        AuthToken token;
}
```

An example of the response message is as following:

```
HTTP/1.1 200 OK
Host: example.com
Content-Length: ***
Content-Type: application/decade-getdata-req+json
X-DECADE-Protocol-Version: 1.0
X-DECADE-TOKEN: DECADE signature
```

8. Remote_Get_Object Message

This operation is used to download objects from a remote DECADE server via another DECADE server. The basic idea is to leverage the base functionality of a "non-transparent proxy" in DECADE. In this model, the local DECADE Server would act as a non-transparent caching proxy (with additional functionality) in order to reach the remote DECADE server and cache the contents of the reply in DECADE storage. A general call flow among the DECADE Client, local DECADE Server, and remote DECADE Server is shown in Figure 2.

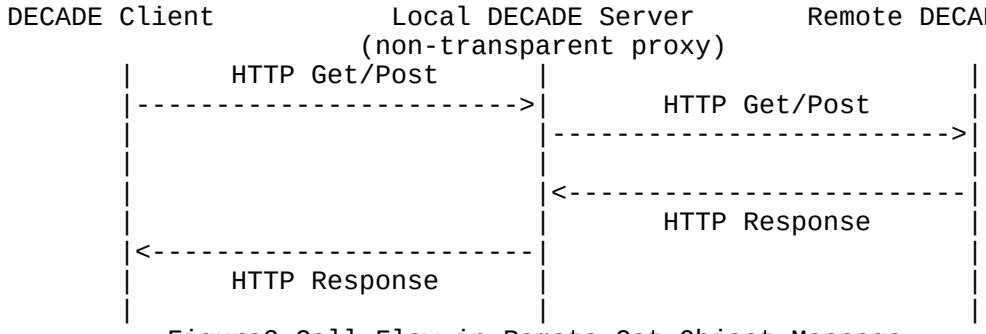


Figure2 Call Flow in Remote_Get_Object Message

If DECADE is using HTTP URI types that existing proxies understand, HTTP-Get should be used. If the non-HTTP URI types are used, it'd better to use POST and to provide DECADE request parameters such as the object name in form data parameters. It can cache responses to those requests using cache-control headers.

As for server-server communications (without a DECADE Client involved), standard HTTP GET, PUT, POST could be used. Figure 3 gives a simple call flow between two DECADE Servers for server-server communications.

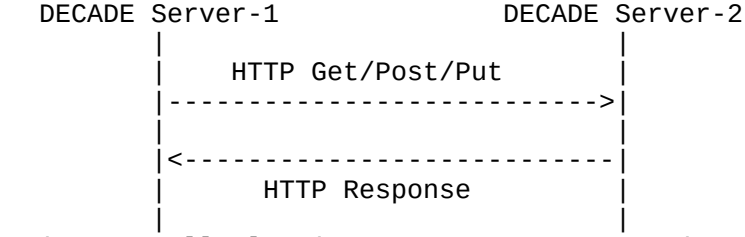


Figure3 Call Flow in Server-Server Communications

9. Security Considerations

The security considerations described in both [I-D.ietf-decade-arch] and [I-D.ieft-decade-problem-statement] apply to this draft as well.

10. IANA Considerations

This draft does not have any IANA considerations.

11. Acknowledgements

We would like to express our sincere thanks and appreciation to Zhou

Hong for his valuable suggestions and support.

12. References

12.1. Normative Reference

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn , "The Kerberos Network Authentication Service (V5)", July 2005.

[RFC2865] Rigney, C., Willens, S., Rubens A., A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", June 2000.

[RFC2866] Rigney, C., "RADIUS Accounting", June 2000.

[RFC3588] Calhoun , P., Loughney , J., Rubens A., E., Zorn , G., and J. Arkko, "Diameter Base Protocol", September 2003.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", June 1999.

12.2. Informative Reference

[I-D.ietf-decade-reqs]
Gu, Y., Bryan, D., Yang, Y., and R. Alimi, "DECADE Requirements", September 2011.

[I-D.ietf-decade-arch]
Alimi , R., Yang, Y., Rahman, A., Kutscher, D., and H. Liu, "DECADE Architecture", July 2011.

[I-D.ietf-decade-problem-statement]
Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement".

[I-D.ietf-oauth-protocol-v2-23]
Recordon, D. and D. Hardt, "The OAuth 2.0 Authorization Protocol", January 2012.

Authors' Addresses

Wang Danhua
Huawei Technologies
No. 101 Software Avenue
Nanjing, Jiangsu Province 210001
P.R.China

Phone: +86-25-56624734
Fax: +86-25-56624702
Email: wangdanhua@huawei.com

Harry Liu
Yale University

Email: hongqiang.liu@yale.edu

Y.Richard Yang
Yale University

Email: yry@cs.yale.edu

DECADE
Internet-Draft
Intended status: Informational
Expires: April 25, 2012

R. Yang
Yale University
N. Zong
Huawei Technologies
October 23, 2011

DECADE Content Replication and Access
draft-yry-decade-replication-00

Abstract

The DECADE Working Group at IETF is working on introducing standard-based, application-agnostic in-network storage for content distribution to improve both network efficiency and application performance. In this document, we introduce content replication trees and then discuss approaches on how a replication tree can be constructed in DECADE. In particular, we introduce concepts including core-stateless content pull, and stateful, open content forwarding table.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2012.

Copyright Notice

Table of Contents

1. Introduction 4
2. Entities and Terms 4
3. DECADE Content Replication Tree Structure 5
3.1. Simple Replication and Access Tree 5
3.2. Recursive Data Replication Tree 6
4. DECADE Content Replication Tree Construction 7
4.1. Proactive Push vs On-Demand Pull 8
4.2. Application Control vs DECADE Control 9
5. On-Demand Pull for DECADE 9
5.1. Core-Stateless: Request Pull Chain 10
5.2. Core-Stateful: Open Content Forwarding Table 10
6. Hybrid Control Approaches 11
7. Security Considerations 11
8. IANA Considerations 11
9. Acknowledgements 11
10. References 11
10.1. Normative References 11
10.2. Informative References 11
Authors' Addresses 12

1. Introduction

The IETF DECADE architecture [I-D.ietf-decade-arch] enables applications to control in-network storage (INS) to achieve efficient content distribution. The DECADE architecture is motivated in the context of P2P applications to save last-mile uplink bandwidth. But DECADE may also be applied to a wider range of applications. See [I-D.ietf-decade-reqs] for a definition of the target applications supported by DECADE.

In particular, DECADE introduces capabilities where applications can have explicit control over writing and reading at specified in-network storage locations. A Standard Data Transport (SDT) protocol [I-D.ietf-decade-arch] is used to write (upload) data objects to a DECADE Server and to read (download) data objects from a DECADE Server.

In this document, we focus on two issues: DECADE Content Replication and DECADE Content Access. DECADE Content Replication is to determine, which DECADE Server(s) should store a given piece of content, when, and from where does a storing DECADE server get the content? The DECADE Content Access is to determine, for a request for content from a Content Consumer, which server should serve the request?

The solution to Content Replication and Access depends on the setting: the workload and the capability of the Content Distribution Application that is utilizing DECADE. The effectiveness of a scheme for in-network storage for content distribution depends on the overhead of pushing content into in-network storage versus the benefit of having the content available into in-network storage.

Since this document is a first step, we consider a basic setting where a single DECADE Service Provider offers to a Content Distribution Application multiple DECADE Servers distributed across a network. We focus on a single piece of content, which can be ingested into DECADE Servers from a Content Provider. Some Content Consumers will consume the piece of content.

2. Entities and Terms

This document uses entities defined in [I-D.ietf-decade-arch], including DECADE Server, DECADE Service Provider, DECADE Content Provider, DECADE Content Consumer, Content Distribution Application, etc.

This document also defines additional terminology:

Application Control Logic: We use this term to refer to the generic data-flow control logic implemented by a Content Distribution Application to control which DECADE Server(s) store the piece of data content, and which DECADE Server serves a request for a piece of data. This logic can be centralized or distributed.

DECADE Control Logic: This is similar to Application Control Logic, but implemented by the DECADE Service Provider.

3. DECADE Content Replication Tree Structure

Since we focus on the data flow of a specific single piece of content, we consider that the Replication and Access data flow forms a tree.

3.1. Simple Replication and Access Tree

We first show a simple Replication and Access structure, as shown in below Figure 1.

We identify three types of content data flow links in the tree:

Content Provider -> Content Consumer;

Content Provider -> DECADE Server;

DECADE Server -> Content Consumer.

The sub-tree formed by omitting the Content Consumer nodes is the Replication tree. Since there are no data flow links among DECADE Servers, the Replication tree is one level.

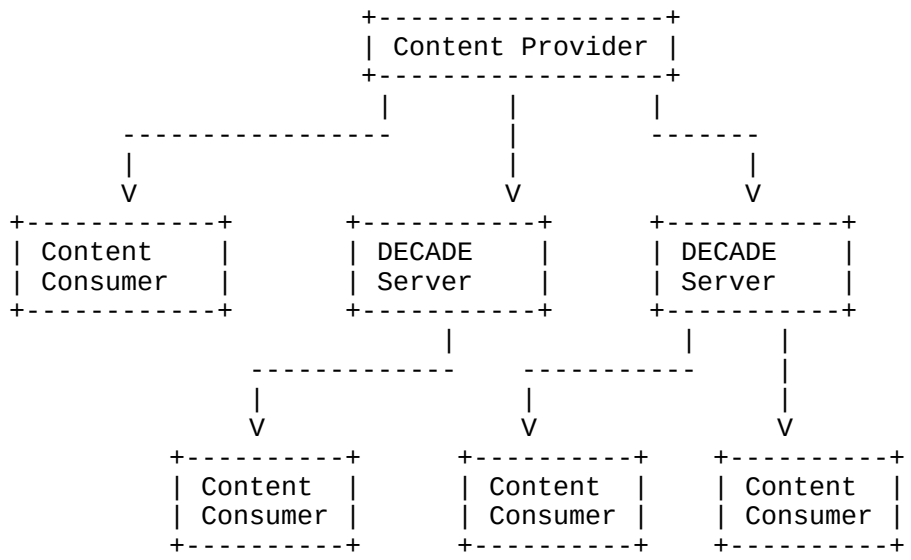


Figure 1: A Simple Replication and Access Tree.

3.2. Recursive Data Replication Tree

A one-level Replication Tree may not be scalable. For example, if there is a flash-crowd arrival of Content Consumers, there may not be enough capacity at the Content Provider to replicate simultaneously to many DECADE Servers to serve many Content Consumers. What this implies is that DECADE must support inter-DECADE server replication, i.e., the capability to replicate the content from one DECADE server to another DECADE server. We refer to a Replication Tree with inter-DECADE Server data flow links as a Recursive Replication Tree. Figure 2 below illustrates an example.

We identify four types of data flow links in the tree:

Content Provider -> Content Consumer;

Content Provider -> DECADE Server;

DECADE Server -> DECADE Server;

DECADE Server -> Content Consumer.

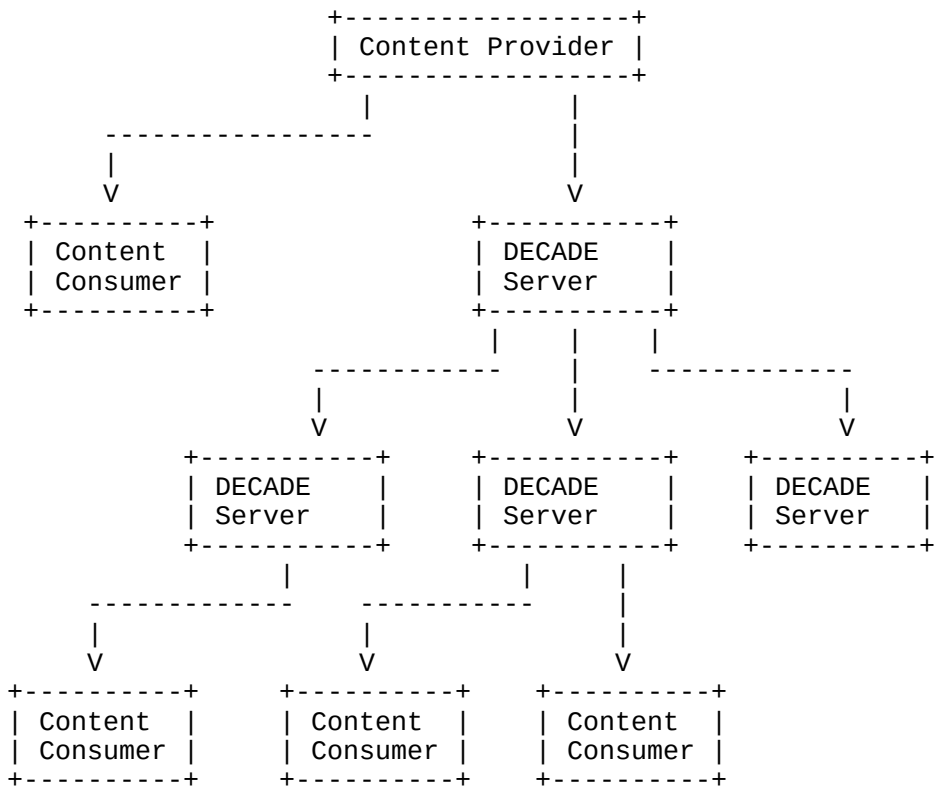


Figure 2: A Recursive Data Replication and Access Tree.

We believe that DECADE should support the construction of Recursive Replication trees.

4. DECADE Content Replication Tree Construction

There are multiple design dimensions in constructing a DECADE Content Replication Tree:

Timing: a tree could be constructed proactively or on-demand;

Controller: Both the Application Control Logic (through a DECADE API) and the DECADE Control Logic (through DECADE Service Provider internal optimization) can be involved in constructing a replication tree to achieve effective content distribution.

Below we discuss the two dimensions in more detail.

4.1. Proactive Push vs On-Demand Pull

We identify the following scenarios where a replication link can be created:

Proactive push: The sender pushes (write/post) a piece of content to a DECADE Server before Consumer requests;

On-demand pull: A DECADE Server pulls content into itself upon Consumer request.

Both Proactive Push and On-Demand Pull have their benefits and issues.

Proactive Push: it may reduce content distribution latency. The push operation can also be scheduled to utilize under-utilized resources. An issue of Proactive Push, however, is that the resource waste ratio can be higher, in particular, if the request patterns of Content Consumers are difficult to predict. On the other hand, if the request patterns can be known approximately, e.g., for patch distribution, this may not be an issue.

On-demand Pull: it can have a much lower ratio of wasted resources. In particular, if a DECADE Server is on or close to the distribution path to a Content Consumer, then the replication into the DECADE Server is essentially piggybacked on the service to the Content Consumer.

To be more concrete, in Table 1 below, we look at how each entity may initiate each given type of content replication link.

	Content Provider -> Content Consumer	Content Provider -> DECADE Server	DECADE Server -> DECADE Server	DECADE Server -> Content Consumer
Content Provider	Push to the Content Consumer. Issue: How does the Content Provider know to push to the Content Consumer?	Push to DECADE Server. Issue: How does the Content Provider select the DECADE Server?	Recursive push. Issue: How does the Content Provider select the link?	Recursive push. Issue: How does the Content Provider select the link?
==========	==========	==========	==========	==========

Content Consumer	Pull from Content Provider	Recursive pull	Recursive pull	Pull from DECADE Server

Table 1: Proactive Push vs On-Demand Pull.

4.2. Application Control vs DECADE Control

From the below Table 2, one can identify extreme control cases: (1) Application Control Logic conducts the full control; (2) DECADE Control Logic provides the full control.

	Content Provider -> Content Consumer	Content Provider -> DECADE Server	DECADE Server -> DECADE Server	DECADE Server -> Content Consumer
Application Control Logic	Send command to sender to push or to receiver to pull.	See left. Issue: How much should Application knows about DECADE servers and request patterns?	See left.	See left.
==========	==========	==========	==========	==========
DECADE Control Logic	Send command to sender to push or receiver to pull.	See left.	See left.	See left.

Table 2: Application Control vs DECADE Control.

5. On-Demand Pull for DECADE

The current DECADE architecture document [I-D.ietf-decade-arch] introduces the method of DECADE GET with a REMOTE_SERVER parameter to obtain data from a remote DECADE Server. One can consider this method as a specific pull-based inter-DECADE Server replication primitive. A challenge of DECADE design is how to support generic pull-based replication.

In this document, we introduce two techniques.

5.1. Core-Stateless: Request Pull Chain

In this design, we introduce the concept of a pull chain in DECADE GET method. An example of such a pull chain is shown in Figure 3:

Figure 3: Pull Chain in DECADE GET.

This pull chain can be used by a DECADE Client or Server to instantiate a request to pull a data object with Object ID. Specifically, if the DECADE Server receiving this request does not have the data, it will pull from NextHop S1, who may pull from NextHop S2 if it does not have, etc.

The construction of the pull chain is implemented by either the Application Control Logic or the DECADE Control Logic.

5.2. Core-Stateful: Open Content Forwarding Table

In this design, we introduce the novel concept of DECADE Content Forwarding Table at DECADE Servers. An example data plane structure of a DECADE Server with Content Forwarding Table is shown in Figure 4:

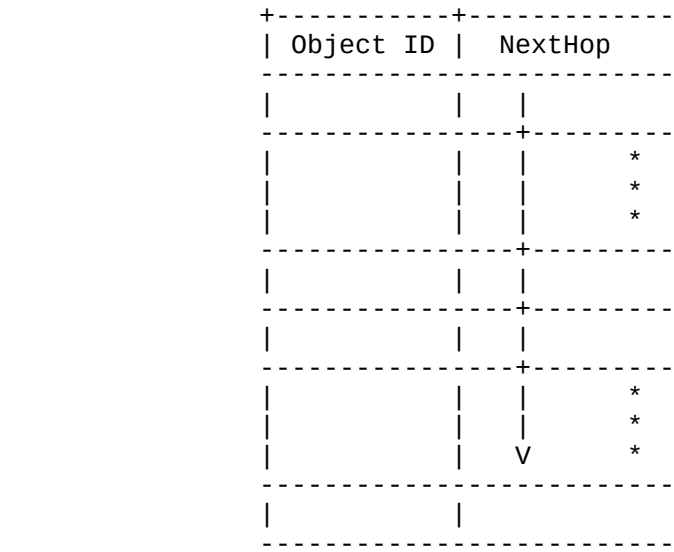


Figure 4: DECADE Data Plane with Open Content Forwarding Table.

This Open Content Forwarding Table is used by DECADE Server to forward the content request based on looking-up the Next Hop of the request. If this is the approach, then DECADE needs to:

GAP 1: Open an interface to write the Content Forwarding Table;

GAP 2: Consider the issue of hierarchical object ID space to allow construction of compact content forwarding table.

6. Hybrid Control Approaches

One design option of DECADE is that a DECADE Provider partitions its DECADE Servers into zones, where each zone has multiple DECADE Servers. One hybrid control mechanism is that the Application Control Logic controls the data replication at the zone level and DECADE Control Logic controls intra zone replication.

7. Security Considerations

TBA

8. IANA Considerations

This document does not have any IANA considerations.

9. Acknowledgements

TBA

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3744] Clemm, G., Reschke, J., Sedlar, E., and J. Whitehead, "Web

Distributed Authoring and Versioning (WebDAV) Access Control Protocol", RFC 3744, May 2004.

[RFC4331] Korver, B. and L. Dusseault, "Quota and Size Properties for Distributed Authoring and Versioning (DAV) Collections", RFC 4331, February 2006.

[RFC4709] Reschke, J., "Mounting Web Distributed Authoring and Versioning (WebDAV) Servers", RFC 4709, October 2006.

[RFC4918] Dusseault, L., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

[I-D.ietf-decade-problem-statement] Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", draft-ietf-decade-problem-statement-04 (work in progress), October 2011.

[I-D.ietf-decade-survey] Alimi, R., Rahman, A., and Y. Yang, "A Survey of In- network Storage Systems", draft-ietf-decade-survey-06 (work in progress), August 2011.

[I-D.ietf-decade-reqs] Yingjie, G., Bryan, D., Yang, Y., and R. Alimi, "DECADE Requirements", draft-ietf-decade-reqs-04 (work in progress), September 2011.

[I-D.ietf-decade-arch] Alimi, R., Yang. Y., Rahman, A., Kutscher, D., and H. Liu, "DECADE Architecture", draft-ietf-decade-arch-03 (work in progress), September 2011.

[GoogleStorageDevGuide] "Google Storage Developer Guide", <http://code.google.com/ apis/storage/docs/developer-guide.html>.

Authors' Addresses

Richard Yang
Yale University

Email: yry@cs.yale.edu

Ning Zong
Huawei Technologies

Email: zongning@huawei.com