DECADE
Internet-Draft
Intended status: Informational
Expires: December 2, 2012

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

May 31, 2012

DECADE Architecture
draft-ietf-decade-arch-06

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 required for introducing a DECADE-compatible 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

Table of Contents

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 DECADE-compatible 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 may 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 a DECADE-compatible 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 supported by a DECADE-compatible system.

The approach of this document is to define the core functionalities and protocol behaviour that are needed to support in-network storage in a DECADE-compatible system. The protocol themselves 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

2.1. DECADE-compatible Client

A DECADE-compatible client uploads and/or retrieves data from DECADE-compatible servers. We simply use the 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 that data. We

simply use the term "server" if there is no ambiguity.

2.3. Content Provider

A client which owns (i.e. uploads and manages) storage at a DECADE-compatible server.

2.4. Content Consumer

A client which has been granted permission to retrieve data from a DECADE-compatible server by a Content Provider.

2.5. Storage Provider

A Storage Provider deploys and/or manages DECADE-compatible server(s) within a network.

2.6. Content Distribution Application

A Content Distribution Application is an application (e.g., P2P) designed for dissemination of a large amounts of data to multiple consumers. Content Distribution Applications typically divide content into smaller blocks for dissemination.

We consider Content Distribution Applications that include a DECADE-compatible client along with other application functionality (e.g., P2P video streaming client).

2.7. Application End-Point

An Application End-Point is an instance of a Content Distribution Application. A particular Application End-Point might be a Content Provider, a Content 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.

2.8. 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.

3. Protocol Flow

3.1. Overview

A DECADE-compatible system will support two logical protocols, as shown in Figure 1. First, the DECADE Resource Protocol (DRP) is responsible for communication of access control and resource scheduling policies between a client and 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.

```
                         Native Application
         .-------------.      Protocol(s)     .-------------.
         | Application | <------------------> | Application |
         |  End-Point  |                      |  End-Point  |
         |             |                      |             |
         | .--------.  |                      | .--------.  |
         | | DECADE |  |                      | | DECADE |  |
         | | Client |  |                      | | Client |  |
         | `--------'  |                      | `--------'  |
         `-------------'                      `-------------'
             |     ^                              |     ^
     DECADE  |     | Standard                     |     |
    Resource |     |   Data                   DRP |     | SDT
    Protocol |     | Transfer                     |     |
     (DRP)   |     |   (SDT)                      |     |
             |     |                              |     |
             |     |                              |     |
             |     |                              |     |
             |     |                              |     |
             |     |                              |     |
             |     |                              |     |
             v     V                              v     V
         .=============.         DRP          .=============.
         |   DECADE    | <------------------> |   DECADE    |
         |   Server    | <------------------> |   Server    |
         `============='         SDT          `============='
```

Figure 1: Generic Protocol Flow

Second, a Standard Data Transfer (SDT) protocol 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 protocols (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. The abstract properties of the SDT and DRP are outlined below but the final selection of these protocols is outside the scope of this document.

3.2. An Example

This section provides an example of steps in 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-compatible 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-compatible 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-compatible storage server, S(A). See Section 6.1.2 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.

```
                                 .----------.
   2. Obtain      --------> |   S(A)    | <------
      Token      /           `----------'        \   4. Request and
      (DRP)     /                                 \    Download Object
      Locally  /                                   \    (DRP + SDT)
      or From /                                     \
      S(A)   v          1. App Request               v
    .-------------. <--------------------------- .-------------.
    | Application |                               | Application |
    | End-Point A |                               | End-Point B |
    `-------------' ---------------------------> `-------------'
                       3. App Response (token)
```

Figure 2: Download from Storage Server

4. Architectural Principles

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

4.1. Decoupled Control/Metadata and Data Planes

A DECADE-compatible 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, ad insertion, 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-compatible 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-compatible 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 blocks that require application level assembly. Many Content Distribution Applications divide bulk content into blocks for multiple reasons, including (1) multipath: different blocks might be fetched from different sources in parallel; and (2) faster recovery and verification: individual blocks might be recovered and verified. Typically, applications use a block size larger than a single packet in order to reduce control overhead.

A DECADE-compatible system SHOULD be agnostic to the nature of the data objects and SHOULD NOT specify a fixed size for them. Though a protocol specification based on this architecture MAY prescribe requirements on minimum and maximum sizes by compliant implementations. Applications may consider existing blocks as data objects, or they may adjust block sizes before storing in the DECADE-compatible server.

Immutable data objects can still be deleted. Applications may support modification of existing data stored at a DECADE-compatible 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 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 data objects/blocks are assumed to be immutable.

4.3. Data Objects With Identifiers

An object that is stored in a DECADE-compatible storage server SHOULD 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-compatible 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.

4.4. Data Object Naming Scheme

The DECADE architecture is based on data object identifiers as described above, and the assignment/derivation of the data object identifier to a data object depends on the data object naming scheme. The details of data naming schemes will be provided by future DECADE-compatible 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-compatible naming scheme follows the following general requirements:

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

o 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);

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

o 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-compatible objects are immutable. But there maybe other applications such as live streaming where object/chunk 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:

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

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

The naming scheme MAY additionally provide the following name elements:

o 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.

The DECADE-compatible naming scheme SHOULD be used in scenarios where a client knows the name of a data object before it is completely stored at the 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 client A might simultaneously begin storing an object to a server, and advertise that the object is available to client B. If it is supported by the server, client B might begin downloading the object before A is finished storing the object.

If object names are not based on hashes of the data objects themselves, names can also be used in scenarios where a client knows the name of a data object before it is locally created.

4.5. 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-compatible 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 store 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).

4.6. Resource and Data Access Control

A DECADE-compatible 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.

4.6.1. Resource Allocation

There are two primary interacting entities in a DECADE-compatible system. First, Storage Providers SHOULD coordinate where storage servers are provisioned and their total available resources. 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 DECADE Content Providers. This means that Content Providers are able to explicitly and independently manage their own shares of resources on a server.

4.6.2. User Delegations

Storage Providers will have the ability to explicitly manage the entities allowed to utilize the resources at a DECADE-compatible 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-compatible 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-compatible system.

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

```
                                    Native Protocol(s)
                            (with other Application End-Points)
                                    .---------------------->
                                    |
                                    |
   .----------------------------------------------------------.
   | Application End-Point                                    |
   | .------------.                 .------------------.     |
   | | App-Layer  |   ...           | App Data Assembly |     |
   | | Algorithms |                 |    Sequencing     |     |
   | `------------'                 `------------------'     |
   |                                                          |
   | .------------------------------------------------------. |
   | | DECADE Client                                        | |
   | |                                                      | |
   | | .-------------------------. .----------------------. | |
   | | | Resource Controller     | | Data Controller      | | |
   | | | .--------. .----------. | | .--------. .-------. | | |
   | | | |  Data  | | Resource | | | |  Data  | | Data  | | | |
   | | | | Access | | Sharing  | | | | Sched. | | Index | | | |
   | | | | Policy | |  Policy  | | | |        | |       | | | |
   | | | '--------' `----------' | | `--------' `-------' | | |
   | | `-------------------------' `----------------------' | |
   | |             |                   ^                    | |
   | `------------ | ----------------- | -------------------' |
   `-------------- | ----------------- | ---------------------'
                   |                   |
                   |  DECADE           | Standard
                   | Resource          |   Data
                   | Protocol          | Transfer
                   |   (DRP)           |   (SDT)
                   v                   V
```

Figure 3: Application Components

5.1.1. Data Assembly

A DECADE-compatible system is geared towards supporting applications that can divide distributed content into 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-compatible system assumes that applications implement this functionality themselves. See Section 5.3 for further discussion.

5.1.2. Native Application Protocols

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

5.1.3. DECADE Client

An application needs to be modified to support a DECADE-compatible system. The client provides the local support to an application, and can 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-compatible 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-compatible 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. Server

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

```
          |                   |
          |  DECADE           | Standard
          | Resource          |   Data
          | Protocol          | Transfer
          |   (DRP)           |   (SDT)
          |                   |
       .= | ================= | ======================.
       |  |                   v                       |
       |  |      .----------------.                   |
       |  |----> | Access Control | <---------.        |
       |  |      `----------------'          |        |
       |  |                   ^              |        |
       |  |                   |              |        |
       |  |                   v              |        |
       |  |   .---------------------.        |        |
       |  `-> | Resource Scheduling | <------|        |
       |      `---------------------'        |        |
       |                      ^              |        |
       |                      |              |        |
       |                      v        .------------. |
       |        .-----------------.    |    User    | |
       |        |    Data Store   |    | Delegation | |
       |        `-----------------'    | Management | |
       | DECADE Server                 `------------' |
       `==============================================='
```

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-compatible 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-compatible server. Data SHOULD be deleted from storage either explicitly or

automatically (e.g., after a TTL expiration). It SHOULD 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-compatible server will only be responsible for storing and retrieving individual data objects. Furthermore, a DECADE-compatible system will use its own naming scheme that provides uniqueness (with high probability) between data objects, even across multiple applications.

5.3.1. Data Object Naming Scheme

Details of the naming scheme are discussed in Section 4.4.

5.3.2. Application Usage

Recall from Section 5.1.1 that an Application typically includes its own naming and sequencing scheme. The DECADE-compatible 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-compatible client may use a naming and sequencing scheme independent of DECADE-compatible names. The DECADE-compatible client SHOULD maintain a mapping from its own data objects and their names to the DECADE-specific data objects and names.

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 16 KiloByte (KB).

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

shown in Figure 5.

```
  Application Chunks
.---------.---------.---------.---------.---------.---------.--------
|         |         |         |         |         |         |
| Chunk_0 | Chunk_1 | Chunk_2 | Chunk_3 | Chunk_4 | Chunk_5 | Chunk_6
|         |         |         |         |         |         |
`---------`---------`---------`---------`---------`---------`--------


  DECADE Data Objects
.---------------------------------------.----------------------------
|                                       |
|               Object_0                |               Object_1
|                                       |
`---------------------------------------`----------------------------
```

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-compatible data object given the chunks contained within that data object. The name might be learned from either the original source, another End-Point with which the Application is communicating, a tracker, etc.

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

5.3.3.2. 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.

```
     Application's Video Stream
   .---------------------------------------------------------------------
   |
   |
   |
   `---------------------------------------------------------------------
   ^              ^              ^              ^              ^
   |              |              |              |              |
   0 Seconds     10 Seconds     20 Seconds     30 Seconds     40 Seconds
   0 B           400 KB         900 KB         1200 KB        1500 KB


     DECADE Data Objects
   .--------------.--------------.--------------.--------------.--------
   |              |              |              |              |
   |   Object_0   |   Object_1   |   Object_2   |   Object_3   |
   |   (400 KB)   |   (500 KB)   |   (300 KB)   |   (300 KB)   |
   `--------------`--------------`--------------`--------------`--------
```

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 Authentication and Resource Control

A key feature of a DECADE-compatible system is that a client can authorize other Clients to store or retrieve data objects from the in-network storage. A token-based authentication scheme is used to accomplish this.

Specifically, an entity trusted by a client generates a signed token with particular properties (see Section 6.1.2 for details). The client then distributes this token to other Clients which then use the token when sending requests to the DECADE-compatible server. Upon receiving a token, the server validates the signature and the operation being performed (e.g. PUT, GET).

This is a simple scheme, but has some important advantages over an alternate 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-compatible target applications:

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

whom they are distributed.

- o 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.

- 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 in other administrative or security domains, and allow them to read or write data objects from its storage.

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 possess a valid token.

Existing protocols (e.g., OAuth [RFC5849]) 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-compatible 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-compatible servers to which a 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.

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-compatible client-server are very similar to the DRP/SDT functionality between running between server-server. Any differences are highlighted below.

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

6.1. DECADE Resource Protocol (DRP)

DRP will provide configuration of access control and resource sharing policies on DECADE-compatible servers. A Content Distribution Application, e.g., a live P2P streaming session, can have permission to manage data at several servers, for instance, servers in different operator domains, 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.

6.1.1. Controlled Resources

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

o Communication resources in terms of bandwidth (upload/download) and also in terms of number of connected clients (connections) at a time.

o Storage resources.

6.1.2. Access and Resource Control Token

A token SHOULD include the following information:

- o Permitted operations (e.g., read, write)
- o Permitted objects (e.g., names of data objects that might be read or written)
- o Expiration time
- o Priority for bandwidth given to requested operation (e.g., a weight used in a weighted bandwidth sharing scheme)
- o Amount of data that might be read or written

The tokens SHOULD be generated by an entity trusted by both the DECADE-compatible client and server at the request of a DECADE-compatible client. For example this entity could be the client, a server trusted by the client, or another server managed by a Storage Provider 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 sending client's server 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 4.6), the a Content Provider should be able to indicate if a token may be used multiple times.

It SHOULD be possible for DRP to allow tokens to apply to a batch of operations to reduce communication overhead required between clients. A request sent in this way explicitly denotes the objects to which it applies.

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.

6.1.3. Status Information

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

Status information on a specific server: Access to such status information SHOULD require client authorization, i.e., clients need to be authorized to access the requested status information. This authorization is based on the user delegation concept as described in Section 4.6. The following status information elements SHOULD be obtained:

* List of associated objects (with properties)

* Resources used/available

The following information elements MAY additionally be 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 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 objects have been distributed.

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 SHOULD based on the delegation concept as described in Section 4.6. The following type of access information elements MAY be requested:

* What objects have been accessed how many times

* Access tokens that a server as seen for a given 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.

6.1.4. Object Attributes

Objects that are stored on a DECADE-compatible 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 4.6. 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 object might be deleted

Object size: In bytes

Media type Labelling of type as per [RFC4288]

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

The 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.

6.2. Standard Data Transfer (SDT) Protocol

A DECADE-compatible server will provide a data access interface, and the SDT will be 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 server always responds to client requests.

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 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.

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.

If supported, a server can accept download requests for an object that is still being uploaded.

The application that originates the objects generates DECADE-compatible object names according to the naming specification in Section 5.3. 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.

6.2.2. Downloading Objects

A client can request named 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 object exists on the server and then token has been validated, the server delivers the requested object in a response message.

If the 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.

6.3. Server-to-Server Protocols

An important feature of a DECADE-compatible system is the capability for one server to directly download 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 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 (e.g. PUT, GET); and
- The Content Provider 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).
- Credentials indicating permission 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 request. Object attributes (see Section 6.1.4) 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. It is this mode of operation that provides substantial savings in uplink capacity. This mode of operation can also be triggered by an administrative/management application outside the architecture.

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 GET operation, the 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 server is to store the object to the remote server using the specified credentials (via a PUT request to the remote server). 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 [I-D.ietf-decade-problem-statement] apply to this document as well.

A DECADE-compatible 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-compatible clients take part in application-specific communication sessions.

The security considerations here focus on threats related to the DECADE-compatible 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-compatible network of servers 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 its 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. A 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-compatible 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: Object Spoofing

In a DECADE-compatible 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-compatible 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 5.3). 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-compatible servers MAY perform name-object binding validation on stored objects, but Application End-Points MUST NOT rely on that. In other forms: 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 this document:

David Bryan

Hongqiang (Harry) Liu

Yingjie Gu

David McDysan

Borje Ohlman

Haibin Song

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.

10.2. Informative References

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

[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-06 (work in progress), May 2012.

[I-D.ietf-decade-reqs] Yingjie, G., Bryan, D., Yang, Y., and R. Alimi, "DECADE Requirements", draft-ietf-decade-reqs-06 (work in progress), March 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-compatible 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.

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