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 a DECADE (DECoupled Application Data Enroute) compliant system. This document presents an architecture for DECADE, discusses the underlying principles, and identifies key functionalities required for introducing in-network storage for these applications. In addition, some examples are given to illustrate these concepts in an informative manner.

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."
Table of Contents

1. Introduction ........................................... 5
2. Functional Entities ................................... 5
   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 ............... 6
   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 ......................... 10
   4.3. Data Object Identifiers ......................... 11
   4.4. DECADE Data Object Naming Scheme ............ 12
   4.5. Explicit Control ................................ 13
   4.6. Resource and Data Access Control through User Delegation ................................... 14
5. System Components ..................................... 15
   5.1. Content Distribution Application ............ 15
   5.2. DECADE Server ................................... 17
   5.3. Data Sequencing and Naming .................... 19
   5.4. Token-based Authentication and Resource Control .......... 22
   5.5. Discovery ......................................... 23
6. DECADE Protocols .................................... 24
   6.1. DECADE Resource Protocol (DRP) ............... 24
   6.2. Standard Data Transfer (SDT) .................. 27
   6.3. Server-to-Server Protocols ..................... 29
7. Security Considerations ............................... 31
   7.1. Threat: System Denial of Service Attacks .... 31
   7.2. DECADE Protocol Security Threats ............ 32
8. IANA Considerations ................................... 33
9. Acknowledgments ....................................... 33
10. References ............................................ 34
    10.1. Normative References .......................... 34
    10.2. Informative References ....................... 34
Appendix A. In-Network Storage Components Mapped to DECADE Architecture ................................... 35
    A.1. Data Access Interface .......................... 35
    A.2. Data Management Operations ................... 35
    A.3. Data Search Capability ......................... 35
    A.5. Resource Control Interface ..................... 36
    A.6. Discovery Mechanism ............................ 36
    A.7. Storage Mode .................................... 36
1. Introduction

Content Distribution Applications (e.g., P2P 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-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 requirements that are needed to support in-network storage in a DECADE-compatible system. The protocol themselves are not selected or designed. Also, if more complex functionalities are needed, they should be layered on top of the DECADE-compatible system in an application specific manner.

Some illustrative examples are given to help the reader understand certain concepts. These examples are purely informational and are not meant to guide or constrain future protocol design or selection.

2. Functional Entities

This section defines the functional entities involved in a DECADE system. Functional entities are 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 be either 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 be either 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.

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

A DECADE-compatible system supports 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 from a DECADE Client to a DECADE Server, as well as between DECADE Servers. A DECADE-compatible system includes exactly one DRP for interoperability and a common format through which these policies can be communicated.
Second, a Standard Data Transport protocol (SDT) is used to transfer data objects to and from a DECADE Server. A DECADE-compatible system may support multiple SDT’s. If there are multiple SDT’s, a negotiation mechanism is used to determine a suitable data transfer protocol between a DECADE Client and DECADE Server.

The two protocols (DRP and SDT) may or may not be separate on the wire. For example, DRP messages may be piggy-backed within some extension fields provided by certain data transfer 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. The abstract properties of the SDT and DRP are outlined but the final selection of these protocols is left to future steps.
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 A’s DECADE-compatible storage server. 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; Section 6.3 details a case when both A and B wish to employ DECADE-compatible Servers.

When an Application End-Point wishes to use its DECADE-compatible storage server, it provides a token 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-compatible Resource Protocol (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 protocol). Finally, B provides the token to $S(A)$ via DRP, and requests and downloads the data object via a Standard Data Transport (SDT).

```
2. Obtain Token --------> [ S(A) ] <-------
     (DRP) /                        \
Locally /                    \    Download Object
or From /                           \ (DRP + SDT)
   v   S(A)   v  1. App Request

[-----------]<---------------------[-----------]
| 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.
4.1. Decoupled Control/Metadata and Data Planes

A DECADE-compatible system will 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. 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 streaming 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 plane 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 [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 contents 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 contents 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 allows effective reuse of data and de-duplication of redundant content.

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 two reasons: (1) multipath: different blocks may be fetched from different 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.

A DECADE-compatible system is agnostic to the nature of the data objects and does 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 content may 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 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 data objects/blocks are referred to as immutable data objects/blocks.

4.3. Data Object Identifiers

Objects that are stored in a DECADE-compatible storage server can be accessed by content consumers via a data object identifier that has been assigned within a certain application context and that is unique within that application context.

A content consumer may be able to 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-compatible storage provider can still be accessed by a single identifier.

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

Data object identifiers for data objects MUST be created by content providers that upload the objects to DECADE servers.

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

4.4. DECADE Data Object Naming Scheme

The DECADE architecture is based on the data object identifier properties as described in Section 4.3, but does neither specify a specific scheme nor specific name-object binding validation functions (for example, hash functions). Different applications will have specific requirements regarding security (for example, cryptographic strength of hash functions), performance (for example, larger static objects vs streaming) etc.

The details of such naming schemes will be provided by DECADE protocol/naming specifications. This document describes the scheme on a semantic level will but specific SDTs and DRPs may use specific representations. The naming scheme meets the following general requirements:

- Different name-object binding validation mechanisms are supported;
- an application (DECADE content provider) can decide what mechanism to use (or to not provide name-object validation -- for example if authenticity and integrity can be ascertained by alternative means);
- simple (digest hash based) name-object binding validation is supported; and
- applications are able to construct unique names (with high probability) without requiring a registry or other forms of coordination; and
o names are self-describing so that a receiving entity (DECADE content consumer) knows what hash function (for example) to use for validating name-object binding.

For most content distribution scenarios, it will be appropriate to derive the name of a data object from the hash over the data object’s content (the raw bytes), which is made possible by the fact that DECADE-compatible objects are immutable. But there may be other applications such as live streaming where object/chunk names are not based on hashes but rather on an enumeration scheme. The DECADE naming scheme also enables those applications to construct unique names.

In order to enable the uniqueness, flexibility and self-describing properties, the DECADE naming scheme provides 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
- (optionally) additional information such as application context or publisher information.

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

The DECADE naming scheme can 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 may 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 may begin downloading the object before A is finished storing the object.

If object names are not based on content hashes, DECADE 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 must be able to know and coordinate which data is stored
at particular servers. Thus, in contrast with content 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 may require that a 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).

4.6. Resource and Data Access Control through User Delegation

A DECADE-compatible system 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.

4.6.1. Resource Allocation

There are two primary interacting entities in a DECADE-compatible system. First, Storage Providers coordinate where storage servers are provisioned and their total available resources. Second, Applications coordinate data transfers amongst available servers and between 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 on a server to applications. It means applications are able to explicitly and independently manage their own shares of resources on a DECADE server.

4.6.2. User Delegations

Storage providers 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 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-compatible 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. 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.

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 may 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.
5.1.1. Data Assembly

A DECADE-compatible system is geared towards supporting 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.

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 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 transfer protocols in addition to employing a data transfer protocol with DECADE-compliant support. This is important for applications that are designed to be highly robust (e.g., if internet servers are unavailable).

5.1.3. DECADE Client

An application needs to be modified to support a DECADE-compatible system. The DECADE Client provides the local support to an application. A DECADE Client need not be embedded into the 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-compatible servers. These policies can be existing policies of applications (e.g., tit-for-tat) 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) 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-compatible Server stores data from Application End-Points, and provides control and access of those data to Application End-Points. A Server is not necessarily a single physical machine, it could also be implemented as a cluster of machines.
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-compatible servers. Application End-Points may also authorize other End-Points to store data. If an access is authorized by an Application End-Point, the Server will provide access.

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 Server itself.

5.2.2. Resource Scheduling

Applications apply resource sharing policies or use a custom policy. 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-compatible 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-compatible Server is only responsible for storing and retrieving individual data objects. Furthermore, a DECADE-compatible system 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 objects are immutable. This scheme has multiple appealing properties:

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

The DECADE-compatible 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 it 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-compatible 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 may 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 may begin downloading the object before A is finished storing the object.

In certain scenarios (e.g., where a client has limited computational power), it may be advantageous to offload the computation of a data object’s name to the 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. The DECADE-compatible naming format does 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.

DECADE-compatible names are not necessarily correlated with the naming or sequencing used by the Application using a DECADE-compatible client. The DECADE-compatible client is expected to maintain a mapping from its own data objects and their names to the DECADE-specific data objects and names. Furthermore, the DECADE-compatible 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 the DECADE-compatible system. 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 store data in DECADE-compatible servers in data objects of size 64KB. To accomplish this, it can map a sequence of 4 chunks into a single object, as shown in
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 endpoint with which the it is communicating, a tracker, etc.

As long as the data contained within each sequence of chunks is unique, the corresponding 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 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 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.
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 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 digitally-signed token with particular properties (see Section 6.1.2 for details). The Client 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.

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-compliant 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 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.

- 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 may 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 will prefer to use existing mechanisms wherever possible.

5.5. Discovery

A DECADE-compatible system includes 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 allows 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 may also result in an error if no such servers can be located.) After discovering one or more servers, a client may 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 will 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, or if desired and needed, as an extension to a DECADE-compatible system.

6. DECADE Protocols

This section presents the DECADE Resource Protocol (DRP) and the Standard Data Transfer (SDT) in terms of abstract protocol interactions that are intended to 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 is 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 is used to send the operations to the 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-compatible servers. A content distribution application, e.g., a live P2P streaming session, may 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 endpoint, 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 may be managed:

communication resources: 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: Servers have limited storage resources.

6.1.2. Access and Resource Control Token

A token includes the following information:

Permitted operations (e.g., read, write)

Permitted objects (e.g., names of data objects that may be read or written)

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-compatible Client. It is out of scope of this document to identify which entity serves this purpose, but examples include the Client itself, a Server trusted by the Client, or another server managed by a Storage Provider trusted by the Client.

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 must be sent along with the operation. The Server validates 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 applies the resource control policies indicated in the token while performing the operation.

Tokens include a unique identifier to allow a Server to detect when a token is used multiple times.

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

6.1.3. Status Information

DRP provides a request service for status information that clients can use to request information from a server.
status information per application context on a specific server:
Access to such status information requires client authorization, i.e., 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 4.6. 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 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 4.6. 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 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-compatible 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 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 are not applied to a server or client to which a data object is copied.

Depending on authorization, 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 4.6. The 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:

Expiration Time: Time at which the object may be deleted

object size: in bytes

Media type

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 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, it can be stored within data objects themselves.

6.2. Standard Data Transfer (SDT)

A DECADE-compatible server provides 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 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.

A server may 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. The naming scheme provides that the name is unique. Clients (as parts of application entities) upload a named object to a server. A 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’.

Method PUT:

Parameters:

   NAME: The naming of the object according to Section 5.3

   OBJECT: The object itself.

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

RESPONSES: The server responds 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 server did not match the provided name (i.e., hash validation failed).

   PERMISSION_DENIED: The 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 client can request named objects from a server. In the following, we provide an abstract specification of the download operation that we name ‘GET METHOD’.

Method GET:

Parameters:

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

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

RESPONSES: The server responds 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 server has not found anything matching the request object name.

PERMISSION_DENIED: The 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 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 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
In this way, a server acts as a proxy for a client, and a client may instantiate requests via that proxy. The operations are 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 may also be triggered by an administrative/management application outside the architecture.

6.3.1. Operational Overview

Server-to-server support is focused on reading and writing data objects between servers. A DECADE-compatible GET or PUT request may supply the following additional parameters:

REMOTE_SERVER: Address of the remote 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 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. Object attributes (see Section 6.1.4) may also be specified in the request to the remote 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 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 may optionally be uploaded from the client or may already exist at the proxying 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 generally 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 may 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 endpoints to be authorized to store and to download objects, and application endpoints can delegate authorization to other application endpoints 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) may 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. DECADE Protocol Security Threats

7.2.1. Threat: Authorization Mechanisms Compromised

A DECADE-compatible system does not require application endpoints to authenticate in order to access a server for downloading objects, since authorization is not based on endpoint 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, or at least the affected application context on 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-compliant system, an application endpoint is referring other application endpoints 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 endpoint effectively receives a different object (compared to what the uploading endpoint provided). As result, the downloading endpoint 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 endpoints to servers), servers themselves (and their storage sub-systems), and the SDT by which the object is uploaded,
distributed and downloaded.

A DECADE-compliant systems fundamental mechanism against object spoofing is name-content 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-content binding, for example using content hashes, with different hash functions (different algorithms, different digest lengths). For those application scenarios where content hashes are not applicable (for example live-streaming) other forms of name-content binding can be used (see Section 5.3. This flexibility also addresses cryptographic algorithm evolvability: hash functions may get deprecated, better alternatives may be invented etc., so that applications can choose appropriate mechanisms meeting their security requirements.

DECADE-compliant servers MAY perform name-content binding validation on stored objects, but application endpoints MUST NOT rely on that. In other forms: application endpoints SHOULD perform name-content 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
10. References

10.1. Normative References


10.2. Informative References


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.

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

A.1. Data Access Interface

Users can read and write objects of arbitrary size through the Client’s Data Controller, making use of a SDT.

A.2. Data Management Operations

Users can move or delete previously stored objects via the Client’s Data Controller, making use of a SDT.

A.3. Data Search Capability

Users 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, End-Points may 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

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 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 may be stored at a Server. Applications may consider existing blocks as data objects, or they may 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
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 not expected that this document will ever become a Standards Track specification.
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.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
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.5. Resource Control Interface ......................... 30
A.6. Discovery Mechanism .............................. 30
A.7. Storage Mode .................................... 30

Appendix B. History ..................................... 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:

![Diagram of protocol flow]

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.

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.
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.
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.

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 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.

![Application’s Video Stream Diagram]

- Application’s Video Stream
- ----------------------------------------
- | ^              ^              ^              ^              ^ |
- | 0 Seconds     10 Seconds     20 Seconds     30 Seconds     40 Seconds |
- | 0 B           400 KiB        900 KiB        1200 KiB       1500 KiB |

![DECADE Data Objects Diagram]

- DECADE Data Objects
- ---------------------
- | Object_0 | Object_1 | Object_2 | Object_3 |
- | (400 KiB) | (500 KiB) | (300 KiB) | (300 KiB) |

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:

- 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.

- 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.

- 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.

- 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 be 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
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 [RFC5649] 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
10. References

10.1. Normative References


10.2. Informative References


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. History

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
Integration Examples of DECADE System
draft-ietf-decade-integration-example-03

Abstract

Decoupled Application Data Enroute (DECADE) system is an in-network storage infrastructure which is still under discussion and standardization process in IETF DECADE WG. 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 September 13, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Table of Contents

1. Introduction ........................................... 5
2. Terminology ........................................... 6
   2.1. INS Server ......................................... 6
   2.2. INS Client .......................................... 6
   2.3. INS Operations ...................................... 6
   2.4. INS System .......................................... 6
   2.5. INS Client API ...................................... 6
   2.6. INS-enabled Application Client ...................... 6
   2.7. INS Service Provider ................................ 6
   2.8. INS Portal .......................................... 6
3. INS Client API ........................................... 6
4. Integration of P2P Live Streaming and INS System ............... 7
   4.1. Integration Architecture .......................... 7
      4.1.1. Data Access Messages ......................... 8
      4.1.2. Control Messages .............................. 8
      4.1.3. Object Naming Scheme ......................... 9
   4.2. Design Considerations ............................. 9
      4.2.1. Improve Efficiency for Each Connection ...... 9
      4.2.2. Reduce Control Latency ....................... 9
5. Integration of P2P File Sharing and INS System ................ 10
   5.1. Integration Architecture .......................... 10
6. Integration of ALTO and INS System for File Distribution .... 12
   6.1. Architecture Design ................................ 13
   6.2. CP Uploading Procedure ............................ 14
   6.3. End User Downloading Procedure .................... 15
7. Test Environment and Settings ................................ 16
   7.1. Test Settings ...................................... 16
   7.2. Test Environment for P2P Live Streaming Example ...... 17
      7.2.1. INS Server ..................................... 17
      7.2.2. P2P Live Streaming Client .................... 17
      7.2.3. Tracker ......................................... 17
      7.2.4. Streaming Source Server ...................... 17
      7.2.5. Test Controller ................................ 18
   7.3. Test Environment for P2P File Sharing Example .......... 18
      7.3.1. INS Server ..................................... 18
      7.3.2. Vuze Client .................................... 18
      7.3.3. Tracker ......................................... 18
      7.3.4. Test Controller ................................ 19
      7.3.5. HTTP Server .................................... 19
      7.3.6. PlanetLab Manager .............................. 19
   7.4. Test Environment for Combined ALTO and INS File
        Distribution System .................................. 19
8. Performance Analysis .................................... 19
   8.1. Performance Metrics ................................ 20
      8.1.1. P2P Live Streaming ............................ 20
8.1.2. P2P File Sharing ........................................... 20
8.1.3. Integration of ALTO and INS System for File
        Distribution ............................................. 20
8.2. Results and Analysis ........................................ 20
  8.2.1. P2P Live Streaming ..................................... 20
  8.2.2. P2P File Sharing ....................................... 21
  8.2.3. Integrated ALTO and INS System for File
          Distribution ............................................. 22
9. Short Conclusion .............................................. 22
10. Security Considerations ...................................... 22
11. IANA Considerations ......................................... 22
12. References .................................................. 22
    12.1. Normative References .................................. 22
    12.2. Informative References ................................ 23
Authors’ Addresses ............................................... 23
1. Introduction

Decoupled Application Data Enroute (DECADE) system is an in-network storage infrastructure which is still under discussion and standardization process in IETF DECADE WG. 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 firstly 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. We only show the feasibility of integrated ALTO and INS system without comparing with other content distribution systems at this time. More information would be provided after more experiments are done in the near future.

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. INS Server

A server to simulate DECADE server defined in [I-D.ietf-decade-arch].

2.2. INS Client

A client to simulate DECADE client defined in [I-D.ietf-decade-arch].

2.3. INS Operations

A set of communications between INS server and INS client to simulate DECADE protocols defined in [I-D.ietf-decade-arch].

2.4. INS System

A system including INS servers, INS clients, and INS operations.

2.5. INS Client API

A set of APIs to enable native application client to utilize INS operations.

2.6. INS-enabled Application Client

An INS-enabled application client includes INS client and native application client communicating through INS client API.

2.7. 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.

2.8. INS Portal

A simulated portal operated by INS service provider to offer applications/end users a portal 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.

- **Get_Object**: Get a data object from an INS server with an authorized token.
- **Put_Object**: Store a data object into an INS server with an authorized token.
- **Delete_Object**: Delete a data object in an INS server explicitly with an authorized token. Note that a data object can also be deleted implicitly by setting a Time-To-Live (TTL) value.
- **Status_Query**: Query current status of an application itself, including listing stored data objects, resource (e.g. storage space) usage, etc.
- **Generate_Token**: Generate an authorization token. The 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. In our P2P live streaming example, the token is generated by INS client. In the example of combining ALTO and INS system, the token is generated by the CP.

4. Integration of P2P Live Streaming and INS System

We integrate an INS client into a P2P live streaming client in order that P2P live streaming application can easily leverage INS system for data transmission.

4.1. Integration Architecture

The architecture of the integration of P2P live streaming application and INS system is shown in Figure 1.
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. It is also compatible with original P2P live streaming signaling messages such as peer discovery, data availability announcement, etc.

4.1.1. 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. A data object is a data transfer unit between the INS server and the application client, whose size can be application-customized according to the variable requirements of performance or sensitive factors (e.g. low latency).

4.1.2. Control Messages

The control protocols used between the native P2P live streaming clients are modified BitTorrent-like protocols. Please refer to [BT] for the detailed description of BitTorrent protocols. Native P2P live streaming client uses BitTorrent-like protocols for meta-data exchanging. On the other hand, INS-enabled P2P live streaming client adds an additional message on top of BitTorrent-like protocols for token distribution. By exchanging the authorization tokens, the application clients can retrieve or store data objects into or from the INS servers.
4.1.3. Object Naming Scheme

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 our example. INS-enabled P2P live streaming client uses the name of the data object as the ID to request and retrieve data.

4.2. 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.

4.2.1. 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.

- Batch Request: In order to fully utilize the connection bandwidth of INS server and reduce the overhead, an application client may combine multiple requests in a single request to INS server.

- 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.

4.2.2. 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.
5. Integration of P2P File Sharing and INS System

We integrate an INS client into Vuze - a BitTorrent based file sharing application client, to leverage INS system for data transmission.

5.1. Integration Architecture

The architecture of the integration of Vuze and INS system is shown in Figure 2.

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.

In our design, INS client inserts itself into the Vuze client by intercepting certain BitTorrent messages, and adjusting their handling to send/receive data using the INS operations instead.

In our 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. We will also support hash-
based naming scheme in next version of our implementation.

5.2. Message Flow

In order for a better comparison, we firstly briefly show the diagram of the native Vuze message exchange, and then show the corresponding diagram including the INS system.

```
+--------+                            +--------+
| Vuze   |                            | Vuze   |
| Client1|                            | Client2|
+--------+                            +--------+
               |                                     |
               | HandShake                           |
               | Azureus HandShake                    |
               | BT_BitField                          |
               | BT_Request                           |
               | BT_Piece                             |
```

Figure 3

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 [Vuze] for the detailed description of Vuze messages.
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.

6. Integration of ALTO and INS System for File Distribution

The objective of ALTO service is to give guidance to applications/end users about which content servers to select in order to optimize the
content downloading performance in an ISP network-friendly way (e.g. reducing bandwidth consumption). The core component of ALTO service is called ALTO server which generates the guidance based on ISP network information. The ALTO protocol conveys such guidance from the ALTO server to the applications/end users. 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 Design

The integrated ALTO and INS system allows CPs to upload files to INS servers, and end users to download files from optimal INS servers suggested by ALTO service. Specifically, three key components are developed as follow.

- INS Servers: operated by an INS service provider to store files from CPs.
- INS Portal: operated by an INS service provider to 1) offer CPs a portal site to upload files; 2) provide ALTO service to direct end users to optimal INS servers to download files.
- CP Portal: operated by a CP to publish the URLs of the uploaded files for end user downloading.

The architecture is as follow.
6.2. 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.

- CP uploads the file to the INS portal site via HTTP_POST message.
- INS portal distributes the file to the dedicated INS servers 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 our 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. We will also support hash-based naming scheme in next version of our implementation.

- When the file is uploaded successfully, CP portal will list the URLs of the file for end use downloading.

### 6.3. 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.

<table>
<thead>
<tr>
<th>End User</th>
<th>CP Portal</th>
<th>INS Portal</th>
<th>ALTO Server</th>
<th>INS Server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP_Get</td>
<td>Token</td>
<td>HTTP_Get</td>
<td>ALTO Req</td>
</tr>
<tr>
<td></td>
<td>&lt;--------</td>
<td></td>
<td>&lt;---------</td>
<td>&lt;--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP_Get</td>
<td>ALTO Resp</td>
<td>ALTO Resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimal INS Server address</td>
<td>Optimal INS Server address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Get Data</td>
<td></td>
<td>&lt;--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Data Object</td>
<td>&lt;--------</td>
</tr>
</tbody>
</table>

**Figure 7**

- 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 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 find the optimal INS
server storing the requested file.

o INS portal returns the optimal INS server address to the end user.

o End user connects to the optimal 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 cloud 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 cloud, 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 Figure 8.

---

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 closest INS server according to its geo-location distance to the INS servers. INS-enabled P2P live streaming clients use their INS servers to upload streaming content to neighbor clients.

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 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 Figure 9.

```
+-------------+        +-------------+
|             |        |             |
|INS Server   |  ...   |INS Server   |
|             |        |             |
+-------------+        +-------------+
     |            |                     |
     |            |                     |
+-------------+ +-------------+     +-------------+  +-----------+
| Vuze        | | Vuze        | ... | Vuze        |  |           |
| Client      | | Client      |     | Client      |--|  Tracker  |
+-------------+ +-------------+     +-------------+  +-----------+
     |            |                     |
     |            |                     |
+-------------+   +-------------+    +-------------+
|  PlanetLab  |   |    Test     |    |             |
|  Manager    |   | Controller  |    | HTTP Server |
+-------------+   +-------------+    +-------------+
```

Figure 9

7.3.1. INS Server

INS servers ran on Amazon EC2.

7.3.2. Vuze Client

Both INS-enabled and native Vuze clients ran on PlanetLab. INS client embedded in Vuze client was automatically loaded and ran after Vuze client start up. Vuze clients were divided into one seeding client and multiple leeches. The seeding client ran at a Window 2003 server.

7.3.3. Tracker

Vuze client provides tracker capability, so we did not deploy our own tracker. Tracker was enabled when making a BitTorrent file. The
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. It 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 leeches 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 has 4G CPU, 2G Memory and 10G Disk. 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. Note that for the example of integrating ALTO and INS systems to support file distribution of CPs, we only show the feasibility of such integration without comparing the performance of our implementation with other content distribution systems.
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.

- Startup delay: The duration from a peer joins the streaming channel to the moment it starts to play.
- Piece missed rate: The number of pieces a peer loses when playing over the total number of pieces.
- Freeze times: The number of times a peer re-buffers during playing.
- 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.

- Download traffic: The total amount of traffic (MByte) representing the network downlink resource usage.
- Upload traffic: The total amount of traffic (MByte) representing the network uplink resource usage.
- 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 only 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 at this time. More comprehensive metrics would be provided after more experiments are done in the near future.

8.2. Results and Analysis

8.2.1. P2P Live Streaming

- 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 Figure 10. 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.

<table>
<thead>
<tr>
<th></th>
<th>Download Traffic</th>
<th>Upload Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS-Enabled Vuze</td>
<td>480MB</td>
<td>12MB</td>
</tr>
<tr>
<td>Native Vuze</td>
<td>430MB</td>
<td>430MB</td>
</tr>
</tbody>
</table>

Figure 10

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%.
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 (e.g. 1000M interface card server can supply bandwidth of 940Mbps at most). Each INS server can support about 400 online users for file downloading simultaneously.

9. Short 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.

Note that for the example of integrating ALTO and INS system to support file distribution of CPs, we only show the feasibility of such integration without comparing with other content distribution systems at this time. More information would be provided after more experiments are done in the near future.

10. Security Considerations

The 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

progress), October 2011.


12.2. Informative References

[BT] "http://www.bittorrent.org"

[Vuze] "http://www.vuze.com"


[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
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

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
Internet-Draft    Integration Examples of DECADE System        July 2012

(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF
Contributions published or made publicly available before November
10, 2008. The person(s) controlling the copyright in some of this
material may not have granted the IETF Trust the right to allow
modifications of such material outside the IETF Standards Process.
Without obtaining an adequate license from the person(s) controlling
the copyright in such materials, this document may not be modified
outside the IETF Standards Process, and derivative works of it may
not be created outside the IETF Standards Process, except to format
it for publication as an RFC or to translate it into languages other
than English.
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.

- **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.

- **Get_Object**: Get a data object from an INS server with an authorization token.

- **Put_Object**: Store a data object into an INS server with an authorization token.

- **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.

- **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.
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.

<table>
<thead>
<tr>
<th>Vuze Client1</th>
<th>INS Client1</th>
<th>INS Client2</th>
<th>Vuze Client2</th>
<th>INS Server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>HandShake</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BT_BitField</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BT_Request</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redirect</td>
<td>Get Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data Object</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT_Piece</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3

- Vuze client1 sends a "BT_Request" message to Vuze client2 to request a data object as usual.
- 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.
- INS client1 receives the "Redirect" message and then sends an INS message "Get Data" to the INS server to request the data object.
- INS server receives the "Get Data" message and sends the requested data object back to INS client1 after the token check.
- 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.

- **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.

- **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.

- **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.

![Architecture Diagram]

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.

![Diagram](image)

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 servers 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.

- 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.

```
<table>
<thead>
<tr>
<th>End User</th>
<th>CP Portal</th>
<th>INS Portal</th>
<th>ALTO Server</th>
<th>INS Server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP Get</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Token</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP Get</td>
<td></td>
<td>ALTO Req</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALTO Resp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimal INS Server address</td>
<td>Get Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data Object</td>
</tr>
</tbody>
</table>
```

Figure 6

- End user visits CP portal web page, and finds the URLs for the desired file.

- 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.

- 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
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.

- Amazon EC2: We setup INS servers in Amazon EC2 platform, including four regions around the world - US east, US west, Europe and Asia.
- 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.
- 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.
- 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.
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.

```
+-----------+    +-----------+
|   INS     |----|   INS     |
|   Server  |    |   Server  |
+-----+-----+    +-----+-----+ Amazon EC2

+--------------------------+------------------+
s/                          
+--------------------------+------------------+
| Vuze Client              | Vuze Client      |
+--------------------------+------------------+ PlanetLab

+--------------------------+------------------+
| HTTP Server              | Tracker          |
+--------------------------+------------------+
| Test Controller          | PlanetLab Manager|
+--------------------------+------------------+

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 Windows 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 your 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 Fedora 13 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.
8.1.2. P2P File Sharing

To measure the performance of a P2P file sharing application, we mainly employed the following three metrics.

- **Download traffic**: The total amount of traffic representing the network downlink resource usage.
- **Upload traffic**: The total amount of traffic representing the network uplink resource usage.
- **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

- **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.

- **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.

- **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.

<table>
<thead>
<tr>
<th></th>
<th>Download Traffic</th>
<th>Upload Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS-Enabled Vuze</td>
<td>480MB</td>
<td>12MB</td>
</tr>
<tr>
<td>Native Vuze</td>
<td>430MB</td>
<td>430MB</td>
</tr>
</tbody>
</table>

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


12.2. Informative References

[VzApp] "http://www.vuze.com"

[VzMsg] "http://wiki.vuze.com/w/Azureus_messaging_protocol"


[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
Abstract

Peer-to-peer (P2P) applications have become widely used on the Internet today and make up a large portion of the traffic in many networks. In P2P applications, one technique for reducing the transit and uplink P2P traffic is to introduce storage capabilities within the network. Traditional caches (e.g., P2P and Web caches) provide such storage, but they are complex (due to explicitly supporting individual P2P application protocols and cache refresh mechanisms) and they do not allow users to manage access to content in the cache. For example, content providers wishing to use in-network storage cannot easily control cache access and resource usage policies to satisfy their own requirements. This document discusses the introduction of in-network storage for P2P applications, and shows the need for a standard protocol for accessing this storage.

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 November 8, 2012.

Copyright Notice
Table of Contents

1. Introduction ....................................... 3
2. Terminology and Concepts .......................... 4
3. The Problems ....................................... 4
   3.1. P2P infrastructural stress and inefficiency .... 4
   3.2. P2P cache: a complex in-network storage ...... 5
   3.3. Ineffective integration of P2P applications ... 6
4. Usage Scenarios .................................... 6
   4.1. BitTorrent ..................................... 6
   4.2. Content Publisher ............................. 7
5. Security Considerations ............................ 8
   5.1. Denial of Service Attacks ..................... 8
   5.2. Copyright and Legal Issues ................... 8
   5.3. Traffic Analysis .............................. 8
   5.4. Modification of Information ................. 8
   5.5. Masquerade .................................... 9
   5.6. Disclosure .................................... 9
   5.7. Message Stream Modification ................. 9
6. IANA Considerations ................................ 9
7. Acknowledgments ................................... 9
8. Informative References ......................... 10
Appendix A. Other Related Work in IETF ............. 11
Authors’ Addresses ................................. 13
1. Introduction

Peer-to-peer (P2P) applications, including both P2P streaming and P2P filesharing applications, make up a large fraction of the traffic in many ISP networks today. One way to reduce bandwidth usage by P2P applications is to introduce storage capabilities in networks. Allowing P2P applications to store and retrieve data from inside networks can reduce traffic on the last-mile uplink, as well as on backbone and transit links.

P2P caches provide in-network storage and have been deployed in some networks. However, the current P2P caching architecture poses challenges to both P2P cache vendors and P2P application developers. For P2P cache vendors, it is challenging to support a number of continuously evolving P2P application protocols, due to lack of documentation, ongoing protocol changes, and rapid introduction of new features by P2P applications. For P2P applications, closed P2P caching systems limit P2P applications from effectively utilizing in-network storage. In particular, P2P caches typically do not allow users to explicitly store content into in-network storage. They also do not allow users to implement control over the content that has been placed into the in-network storage.

P2P applications suffer decreased efficiency, and the network infrastructure suffers increased load because there is no standardized interface for accessing storage and data transport services in the Internet.

Both of these challenges can be effectively addressed by using an open, standard protocol to access in-network storage [Data_Lockers]. P2P applications can store and retrieve content in the in-network storage, as well as control resources (e.g., bandwidth, connections) consumed by peers in a P2P application. As a simple example, a peer of a P2P application may upload to other peers through its in-network storage, saving its usage of last-mile uplink bandwidth.

In this document, we distinguish between two functional components of the native P2P application protocol: signaling and data access. Signaling includes operations such as handshaking and discovering peer and content availability. The data access component transfers content from one peer to another.

In essence, coupling of the signaling and data access makes in-network storage very complex to support various application services. However, these applications have common requirements for data access, making it possible to develop a standard protocol.
2. Terminology and Concepts

The following terms have special meaning in the definition of the in-network storage system.

in-network storage: A service inside a network that provides storage and bandwidth to network applications. In-network storage may reduce upload/transit/backbone traffic and improve network application performance. The position of in-network storage is in the core of a network, for example, co-located with the border router (network attached storage) or inside a data center.

P2P cache (Peer to Peer cache): A kind of in-network storage that understands the signaling and transport of specific P2P application protocols. It caches the content for those specific P2P applications in order to serve peers and reduce traffic on certain links.

3. The Problems

The emergence of peer-to-peer (P2P) as a major network application (especially P2P file sharing and streaming) has led to substantial opportunities. The P2P paradigm can be utilized to design highly scalable and robust applications at low cost, compared to the traditional client-server paradigm.

However, P2P applications also face substantial design challenges. A particular problem facing P2P applications is the additional stress that they place on the network infrastructure. Furthermore, lack of infrastructure support can lead to unstable P2P application performance during peer churns and flash crowds, when a large group of users begin to retrieve the content during a short period of time. A potential way to solve it would be to make it possible for peers that were on bandwidth constrained access to put things in a place that is both not bandwidth constrained and accessible by other peers. These problems are now discussed in further detail.

3.1. P2P infrastructural stress and inefficiency

A particular problem of the P2P paradigm is the stress that P2P application traffic places on the infrastructure of Internet service providers (ISPs). Multiple measurements (e.g., [Internet_Study_2008-2009]) have shown that P2P traffic has become a major type of traffic on some networks. Furthermore, the inefficiency of network-agnostic peering (at the P2P transmission level) leads to unnecessary traversal across network domains or spanning the backbone of a network [RFC5693].
Using network information alone to construct more efficient P2P swarms is not sufficient to reduce P2P traffic in access networks, as the total access upload traffic is equal to the total access download traffic in a traditional P2P system. On the other hand, it is reported that P2P traffic is becoming the dominant traffic on the access networks of some networks, reaching as high as 50-60% on the downlinks and 60-90% on the uplinks ([DCIA], [ICNP], [ipoque.P2P_survey.], [P2P_file_sharing]). Consequently, it becomes increasingly important to reduce upload access traffic, in addition to cross-domain and backbone traffic.

The inefficiency is also represented when traffic is sent upstream as many times as there are remote peers interested in getting the corresponding information. For example, the P2P application transfer completion times remain affected by potentially (relatively) slow upstream transmission. Similarly, the performance of real-time P2P applications may be affected by potentially (relatively) higher upstream latencies.

3.2. P2P cache: a complex in-network storage

An effective technique to reduce P2P infrastructural stress and inefficiency is to introduce in-network storage. The existing in-network storage systems can be found in [RFC6392].

In the current Internet, in-network storage is introduced as P2P caches, either transparently or explicitly as a P2P peer. To provide service to a specific P2P application, the P2P cache server must support the specific signaling and transport protocols of the specific P2P application. This can lead to substantial complexity for the P2P Cache vendor.

First, there are many P2P applications on the Internet (e.g., BitTorrent, eMule, Flashget, and Thunder for file sharing; Abacast, Kontiki, Octoshape, PPLive, PPStream, and UUSee for P2P streaming). Consequently, a P2P cache vendor faces the challenge of supporting a large number of P2P application protocols, leading to product complexity and increased development cost.

Furthermore, a specific P2P application protocol may evolve continuously, to add new features or fix bugs. This forces a P2P cache vendor to continuously update to track the changes of the P2P application, leading to product complexity and increased costs.

Third, many P2P applications use proprietary protocols or support end-to-end encryption. This can render P2P caches ineffective. So these three problems make the P2P cache as a network middle-box, hard to support these P2P application distribution in their own ways.
Finally, a P2P cache is likely to be much better connected to end hosts than remote peers that connected to end hosts. Without the ability to manage bandwidth usage, the P2P cache may increase the volume of download traffic, which runs counter to the reduction of upload access traffic.

3.3. Ineffective integration of P2P applications

As P2P applications evolve, it has become increasingly clear that usage of in-network resources can improve user experience. For example, multiple P2P streaming systems seek additional in-network resources during a flash crowd, such as just before a major live streaming event. In asymmetric networks when the aggregated upload bandwidth of a channel cannot meet the download demand, a P2P application may seek additional in-network resources to maintain a stable system.

However, some P2P applications using in-network infrastructural resources require flexibility in implementing resource allocation policies. A major competitive advantage of many successful P2P systems is their substantial expertise in how to most efficiently utilize peer and infrastructural resources. For example, many live P2P systems have specific algorithms to select those peers that behave as stable, higher-bandwidth sources. Similarly, the higher-bandwidth sources frequently use algorithms to choose to which peers the source should send content. Developers of these systems continue to fine-tune these algorithms over time.

To permit developers to evolve and fine-tune their algorithms and policies, the in-network storage should expose basic mechanisms and allow as much flexibility as possible to P2P applications. This conforms to the end-to-end systems principle and allows innovation and satisfaction of specific business goals. Existing techniques for P2P application in-network storage lack these capabilities.

4. Usage Scenarios

Usage scenarios are presented to illustrate the problems in both Content Distribution Network (CDN) and P2P scenarios.

4.1. BitTorrent

When a BitTorrent client A uploads a block to multiple peers, the block traverses the last-mile uplink once for each peer. And after that, the peer B who just received the block from A also needs to upload through its own last-mile uplink to others when sharing this block. This is not an efficient use of the last-mile uplink. With
in-network storage server however, the BitTorrent client may upload
the block to its in-network storage. Peers may retrieve the block
from the in-network storage, reducing the amount of data on the last-
mile uplink. If supported by the in-network storage, a peer can also
save the block in its own in-network storage while it is being
retrieved; the block can then be uploaded from the in-network storage
to other peers.

As previously discussed, BitTorrent or other P2P applications
currently cannot explicitly manage which content is placed in the
existing P2P caches, nor access and resource control policies.
Applications need to retain flexibility to control the content
distribution policies and topology among peers.

4.2. Content Publisher

Content publishers may also utilize in-network storage. For example,
consider a P2P live streaming application. A Content Publisher
typically maintains a small number of sources, each of which
distributes blocks in the current play buffer to a set of the P2P
peers.

Some content publishers use another hybrid content distribution
approach incorporating both P2P and CDN modes. As an example,
Internet TV may be implemented as a hybrid CDN/P2P application by
distributing content from central servers via a CDN, and also
incorporating a P2P mode amongst endhosts and set-top boxes. In-
network storage may be beneficial to hybrid CDN/P2P applications as
well to support P2P distribution and to enable content publisher
standard interfaces and controls.

However, there is no standard interface for different content
publishers to access in-network storage. One streaming content
publisher may need the existing in-network storage to support
streaming signaling or such capability, such as transcoding
capability, bitmap information, intelligent retransmission, etc,
while a different content publisher may only need the in-network
storage to distribute files. However it is reasonable that the
application services are only supported by content publisher’s
original servers and clients, and intelligent data plane transport
for those content publishers are supported by in-network storage.

A content publisher also benefits from a standard interface to access
in-network storage servers provided by different providers. The
standard interface must allow the content publisher to retain control
over content placed in their own in-network storage, and grant access
and resources only to the desired endhosts and peers.
In the hybrid CDN/P2P scenario, if only the endhosts can store content in the in-network storage server, the content must be downloaded and then uploaded over the last-mile access link before another peer may retrieve it from a in-network storage server. Thus, in this deployment scenario, it may be advantageous for a content publisher or CDN provider to store content in in-network storage servers.

5. Security Considerations

There are several security considerations to the in-network storage.

5.1. Denial of Service Attacks

An attacker can try to consume a large portion of the in-network storage, or exhaust the connections of the in-network storage through a Denial of Service (DoS) attack. Authentication, authorization and accounting mechanisms should be considered in the cross domain environment. Limitation of access from an administrative domain sets up barriers for content distribution.

5.2. Copyright and Legal Issues

Copyright and other laws may prevent the distribution of certain content in various localities. In-network storage operators may adopt system-wide ingress or egress filters to implement necessary policies for storing or retrieving content, and applications may apply DRM to the data stored in the network storage. However, the specification and implementation of such policies (e.g., filtering and DRM) is not in scope for the problem this document proposes solving.

5.3. Traffic Analysis

If the content is stored in the provider-based in-network storage, there may be a privacy risk that the provider can correlate the people who are accessing the same data object using the same object identity. This correlation can be used to presume that they have the same interest, so as to use it as a basis for a phishing attack.

5.4. Modification of Information

The modification threat is the danger that some unauthorized entity may alter in-transit in-network storage access messages generated on behalf of an authorized principal in such a way as to effect unauthorized management operations, including falsifying the value of an object. This threat may result in false data being supplied
either through the data on a legitimate store being modified, or through a bogus store being introduced into the network.

5.5. Masquerade

A type of threat action whereby an unauthorized entity gains access to a system or performs a malicious act by illegitimately posing as an authorized entity. In the context of this spec, when accessing in-network storage, one malicious end host can try to act as another authorized end host or application server to access a protected resource in the in-network storage.

5.6. Disclosure

The disclosure threat is the danger of eavesdropping on the exchanges between in-network storage and application clients. Protecting against this threat may be required as a matter of application policy.

5.7. Message Stream Modification

The message stream modification threat is the danger that messages may be maliciously re-ordered, delayed or replayed to an extent which is greater than can occur through natural network system, in order to effect unauthorized management operations to in-network storage. If the middle box, such like NAT (network address translator) or proxy between an end host and in-network storage is compromised, it is easy to do the stream modification attack.

6. IANA Considerations

There are no IANA considerations in this document.

7. Acknowledgments

We would like to thank the following people for contributing to this document:

David Bryan
Ronald Bonica
Kar Ann Chew
Roni Even
8. Informative References


[I-D.ietf-p2psip-base]


[ipoque.P2P_survey.]

[P2P_file_sharing]

[Octoshape]
"http://www.octoshape.com/?page=company/about".


Appendix A. Other Related Work in IETF

(To the RFC editor: Please remove this section and the related references in this section upon publication. The purpose of this section is to give the IESG and RFC editor a better understanding of the current P2P related work in IETF and the relationship with DECADE WG.)

Note that DECADE WG’s work is independent of current IETF work on P2P. The ALTO work is aimed for better peer selection and the RELOAD [I-D.ietf-p2psip-base] protocol is used for P2P overlay maintenance and resource discovery.

The Peer to Peer Streaming Protocol effort in the IETF is investigating the specification of signaling protocols (called the PPSP tracker protocol and peer protocol) for multiple entities (e.g., intelligent endpoints, caches, content distribution network nodes, and/or other edge devices) to participate in P2P streaming systems in both fixed and mobile Internet. As discussed in the PPSP problem statement, one important PPSP use case is the support of an in-
network edge cache for P2P Streaming. However, this approach to providing in-network cache has different applicability, different objectives and different implications for the in-network cache operator. The goal of DECADE WG is to provide in-network storage service that can be used for any application transparently to the in-network storage operator: it can be used for any P2P Streaming application (whether it supports PPSP protocols or not), for any other P2P application, and for non P2P applications that simply want to benefit from in-network storage. With DECADE, the operator is providing a generic in-network storage service that can be used by any application without application involvement or awareness by the operator; in the PPSP cache use case, the cache operator is participating in the specific P2P streaming service.

DECADE and PPSP can both contribute independently, and (where appropriate) simultaneously, to making content available closer to peers. Here are a number of example scenarios:

A given network supports DECADE in-network storage, and its CDN nodes do not participate as PPSP Peers for a given "stream" (e.g., because no CDN arrangement has been put in place between the content provider and the particular network provider). In that case, PPSP Peers will all be "off-net" but will be able to use DECADE in-network storage to exchange chunks.

A given network does not support DECADE in-network storage, and (some of) its CDN nodes participate as PPSP Peers for a given "stream" (e.g., say because an arrangement has been put in place between the content provider and the particular network provider). In that case, the CDN nodes will participate as in-network PPSP Peers. The off-net PPSP Peers (i.e., end users) will be able to get chunks from the in-network CDN nodes (using PPSP protocols with the CDN nodes).

A given network supports DECADE in-network storage, and (some of) its CDN nodes participate as PPSP Peers for a given "stream" (e.g., because an arrangement has been put in place between the content provider and the particular network provider). In that case, the CDN nodes will participate as in-network PPSP Peers. The off-net PPSP Peers (i.e., end users) will be able to get chunks from the in-network CDN nodes (using PPSP protocols with the CDN nodes) as well as be able to get chunks / share chunks using DECADE in-network storage populated by PPSP Peers (both off-net end-users and in-network CDN Nodes).

PPSP and DECADE jointly provide P2P streaming service for heterogeneous networks including both fixed and mobile connections and enables the mobile nodes to use DECADE. In this case there
may be some solutions that require more information in PPSP tracker protocol, e.g., the mobile node can indicate its DECADE in-network proxy to the PPSP tracker and the following requesting peer can finish data transfer with the DECADE proxy.

An ALTO (Application Layer Traffic Optimization) server provides P2P applications with network information so that they can perform better-than-random initial peer selection [RFC5693]. However, there are limitations on what ALTO can achieve alone. For example, network information alone cannot reduce P2P traffic in access networks, as the total access upload traffic is equal to the total access download traffic in a traditional P2P system. Consequently, it becomes increasingly important to complement the ALTO effort and reduce upload access traffic, in addition to cross-domain and backbone traffic.

The IETF Low Extra Delay Background Transport (LEDBAT) Working Group is focusing on techniques that allow large amounts of data to be consistently transmitted without substantially affecting the delays experienced by other users and applications. It is expected that some P2P applications would start using such techniques, thereby somewhat alleviating the perceivable impact (at least on other applications) of their high volume traffic. However, such techniques may not be adopted by all P2P applications. Also, when adopted, these techniques do not remove all inefficiencies, such as those associated with traffic being sent upstream as many times as there are remote peers interested in getting the corresponding information. For example, the P2P application transfer completion times remain affected by potentially (relatively) slow upstream transmission. Similarly, the performance of real-time P2P applications may be affected by potentially (relatively) higher upstream latencies.

Authors’ Addresses

Haibin Song
Huawei
101 Software Avenue, Yuhua District,
Nanjing, Jiangsu Province 210012
China

Phone: +86-25-56624792
Email: haibin.song@huawei.com
Ning Zong
Huawei
101 Software Avenue, Yuhua District,
Nanjing, Jiangsu Province  210012
China

Phone: +86-25-56624760
Email: zongning@huawei.com

Y. Richard Yang
Yale University
Email: yry@cs.yale.edu

Richard Alimi
Google
Email: ralimi@google.com
Abstract

This document explores 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, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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
9. Security Considerations .................................. 19
10. IANA Considerations .................................... 20
11. Acknowledgments ....................................... 20
12. References ........................................... 20
   12.1. Normative Reference ............................. 20
   12.2. Informative Reference ........................... 20
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

```
+-----+      DRP       +-----+
|DECADE|<-------------|DECADE|
|Server|<-------------|Server|
+-----+      SDT       +-----+
        ^  ^                     ^  ^
DRP    |  SDT              DRP    |  SDT
        v  v                     v  v
         ^                        ^
         |                        |
         | DECADE                  | DECADE
         +------------------------+------------------------+

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 status 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 achieve 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
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. However, 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 signature is calculated based
on HMAC-SHA1.

\[
\text{signature} = \text{Base64}(\text{HMAC-SHA1(UTF-8-Encoding-Of(StringToSign))})
\]

\[\text{StringToSign} = \text{PermittedOperation}_1 + \text{"\n"} + \ldots + \text{PermittedOperation}_m + \text{\"\n\"} + \text{PermittedObject}_1 + \text{\"\n"} + \ldots + \text{PermittedObject}_n + \text{\"\n\"} + \text{TTL} + \text{\"\n\"} + \text{Priority for bandwidth given to requested operation} + \text{\"\n\"} + \text{Amount of data that may be read or written} + \text{\"\n\"} + \text{key}\]

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-reg+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;
   JSONObject expiration-time;
   JSONObject 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{
    JSONObject object-type; [OPTIONAL]
    JSONNumber object-size; [OPTIONAL]
    JSONObject 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{
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


This operation is used to download objects from a remote DECADE server via another DECADE server. The only difference between Remote_Get_Object and Get_Object is that in the request of the former, the client should indicate the address of the remote DECADE server with X-DECADE-ORIGIN header. For example, "X-DECADE-ORIGIN: origin.com" header in the request shows it is a Remote_Get_Object request and the remote DECADE server is "origin.com".

9. Security Considerations

The security considerations described in both [I-D.ietf-decade-arch] and [I-D.ietf-decade-problem-statement] apply to this draft as well.
10. IANA Considerations

This draft does not have any IANA considerations.

11. Acknowledgments

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


12.2. Informative Reference

[I-D.ietf-decade-reqs]

[I-D.ietf-decade-arch]

[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

Harry Liu
Yale University
Email: hongqiang.liu@yale.edu

Y.Richard Yang
Yale University
Email: yry@cs.yale.edu