DECADE
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

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

March 14, 2011

Integration Examples of DECADE System
draft-chen-decade-intgr-livestr-exmp-01

Abstract

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

Table of Contents

1. Introduction

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

This draft introduces two instances of application integration with DECADE. In our example system, the core component includes DECADE server and DECADE-aware P2P clients (live streaming client and file-sharing client). A DECADE server runs at Linux platform and is designed to support interactive control and data transport for DECADE clients. For live streaming case, we deployed a P2P live streaming system called P2PLS (P2P Live Streaming). We also utilized a preliminary API (Application Programming Interface) set, which is supposed to be provided by DECADE, to enable P2PLS clients to leverage DECADE in their data transmission. For file-sharing case, we choose an open source P2P client software named Vuze which supports user defining plugin to extend software functions. In this draft, we introduce the structures of the both DECADE integration applications, the DECADE related control flow, the environment of tests on both integrations, and the system performance in the tests.

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

2. Concepts

2.1. P2P

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

2.2. DECADE Server

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

space for applications.

2.3. DECADE Module

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

2.4. P2P LiveStreaming Client (P2PLS Client)

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

2.5. DECADE Client

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

2.6. Vuze

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

2.7. DECADE Plugin

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

2.8. DECADE-Enabled Vuze

A Vuze client that is enabled by DECADE plugin.

2.9. Remote Controller

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

3. DECADE Client API

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

There are five basic interfaces:

- Get_Object: to get an object from a DECADE server with an authorized token. The get operation can be classified into two categories: Local Get and Remote Get. Local Get is to get an object from a local DECADE server. Remote Get is to use a client's local DECADE server to indirectly get an object from a remote DECADE server. The object will be firstly passed to the local DECADE server, then returned to the application client

- Put_Object: to store an object into a DECADE server with an authorized token. A client can either store an object into its local DECADE server or into other clients' DECADE servers, if it has an authorized token. Both of the operations are direct, for we don't provide indirectly storing (i.e. remote put).

- Delete_Object: to delete an object in a DECADE server explicitly with an authorized token. Note that an object can also be deleted implicitly by setting an expired time or a specific TTL.

- Status_Query: to query current status of an application itself, including listing stored objects, resource usage, etc.. Such information is private.

- Generate_Token: to generate an authorized token. The token can be used to access an application client's local DECADE server, or passed to other clients to allow them to access the client's local DECADE server.

4. DECADE Integration of P2P LiveStreaming Client

We integrate a DECADE module into a P2P live streaming application--P2PLS, in order that clients of this application can easily leverage DECADE in their data distributions.

4.1. DECADE Integration Architecture

The architecture of the P2PLS application and DECADE integration is shown in Figure 1:

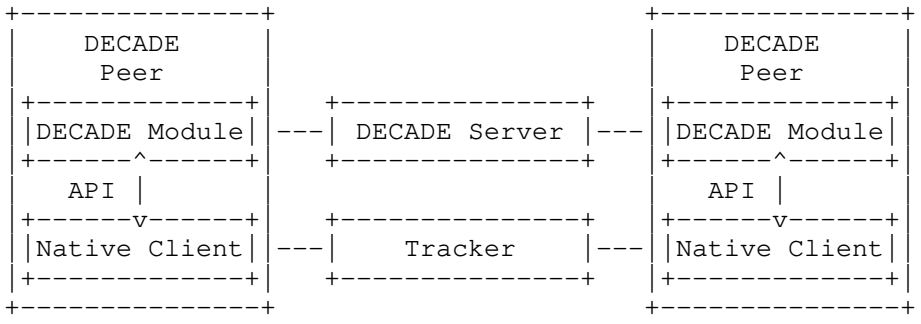


Figure 1

A DECADE-integrated P2PLS client uses DECADE module to communicate with its DECADE server and transmit data between itself and its DECADE server. It is compatible with its original P2P protocol, while it also uses a DECADE protocol to exchange DECADE related messages with other peers.

4.1.1. Data Access

DECADE module is called whenever a client wants to get data objects from (or put data objects into) its DECADE server. Each data object transferred between a client and its DECADE server should go through DECADE module. Neither the DECADE server and the original client knows each other. A data object is a data transfer unit between DECADE servers and application clients. Its size can be application-customized, according to variable requirements of performance or sensitive factors (e.g. low latency, high bandwidth utilization).

4.1.2. Message Control

Control and data plane decoupling is a design principle of DECADE. Control messages are propagated in an original P2P way. DECADE only introduces an additional control message between DECADE module which carries DECADE authorized token. By exchanging DECADE authorized tokens, P2P live streaming clients can retrieve or store data objects into or from others' DECADE servers.

4.2. Challenges in DECADE Integration

One essential objective of DECADE integration is to improve (or at least not to hurt) the application performance. However, as a brand new architecture, DECADE has some inherit challenges which will potentially be harmful to application performance. In our P2P live streaming case, we met mainly two such limitations of DECADE:

4.2.1. Limited Connection Slot

Limited Connection Slot: In native P2P systems, a peer can establish tens or hunderds of concurrent connections with other peers. However, this situation can hardly be ture when it is integrated with DECADE because it is too expensive for DECADE servers to maintain so many connections for each peer. Typically, each DECADE peer only have m connection slots, which means it can only at most have m active connections with its DECADE server simultaneously. A potential "side effect" of limited connection slot is that the content availability and downloading rate might be impacted negatively by fewwer connections carrying data traffic. This requests us to adjust the peer's behavior in resource allocation, downloading/uploading scheduling, etc. to fully utilize the connection slots to archieve a satisfying and robust data downloading.

- Batch Request: In order to fully utilize the connection bandwidth of a DECADE server and reduce overhead, a P2PLS client may combine multiple requests in a single request to DECADE server. Note that for the sake of improving data transfer efficiency in P2P live streaming, we may combine multiple data requests into a batch request. Generally, a batch may consist of different kinds of access requests.

- Data Object Size: In typical P2P live streaming application, the size of a data block is relatively small, considering to reduce end-to-end transport latency. However, existing data size may incur large control overhead and low transport utilization. A larger data object size may be needed to utilize DECADE more efficiently.

4.2.2. Additional Control Latency

Addtional Control Latency: In native P2P systems, when an uploader decides to reply a request for a piece, it sends the piece out directly. Nevertheless, in DECADE-aware P2P systems, the uploader typically only replies with a token of the piece. And then the downloader will leverage this token to fetch the piece from the uploader's DECADE server. This process obviously introduces

additional control latency compared with native P2P systems. It is even more serious in latency sensitive applications such as P2P live streaming. We need to consider how to reduce such additional delay or how to compensate the lose with other inherit advantages of DECADE.

- Range Token: One way to reduce request latency is to use range token. A P2PLS client may piggyback a range token when it propagates its bitmap to its neighbors, to indicate that all available pieces in the bitmap are accassable by this range token. Then instead of requesting specific pieces from this client and waiting for response, the neighbors can directly use this range token to access data in DECADE servers. Note that this method not only reduce request latency, but also reduce message overhead.

With these adjustments and strategies, DECADE clients' performance was not impacted by the liminations of DECADE and was even better than native clients' as we will see in following sections.

5. DECADE Integration of P2P Filesharing Client

We integrate DECADE with a popular P2P file-sharing application-- Vuze.

5.1. Vuze Client Design

Note that Vuze client is classified into two different kinds - Native Vuze and DECADE-Enabled Vuze. When running Native Vuze, it behaves as ordinary BitTorrent client: some Vuze clients upload data for others to download. When using DECADE-Enabled Vuze, the communication and data exchange processes are changed. The uploader uploads data to its DECADE server, and downloaders download data from DECADE servers. By this means, uplink traffic can be reduced sharply and download performance can be improved. It is beneficial for ISPs to save the last-mile uplink bandwidth.

5.2. DECADE-Enabled Vuze architecture Design

DECADE plugin is a key component of our demo system. It has several interfaces with other components as following:

Interface between DECADE plugin and DECADE server: DECADE plugin can upload and download data from DECADE server. It also includes other functions such as user registration, application registration etc.

Interface between DECADE plugin and Vuze client: DECADE plugin can register a listener to intercept the BitTorrent message such as

"BT_Request" message from or to Vuze client, encapsulate the data from DECADE server into "BT_Piece" message and put the "BT_Piece" message into incoming message queue, read the block/piece data from the disk when seeding (to upload the data to DECADE server), and start or stop the download tasks, all these functions are supported in the Plugin API provided by Vuze.

Interface between DECADE plugins: When DECADE plugin intercepts the "BT_Request" message from other Vuze clients, local DECADE plugin sends "Redirect" message to remote DECADE plugin to authorize it to download the piece data from the DECADE server.

Interface between DECADE plugin and Remote Controller: DECADE plugin registers with Remote Controller after starting up, Remote Controller can control all the Vuze clients to start, stop or resume the download tasks through DECADE plugins.

The system architecture is as follow:

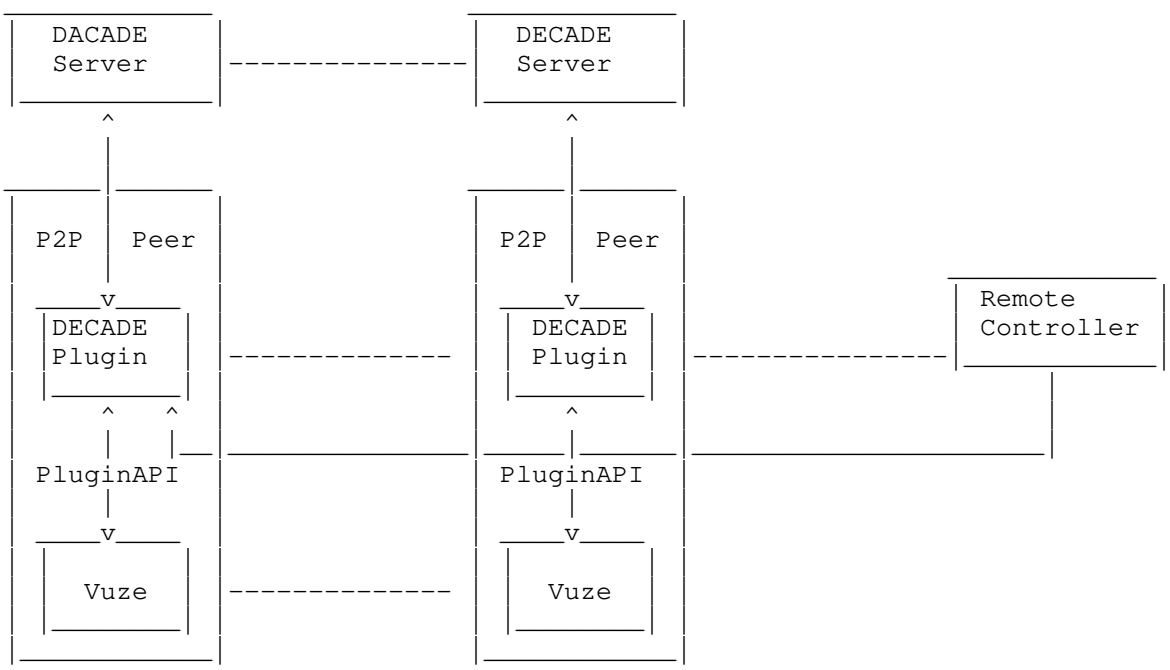


Figure 2

5.3. DECADE-Enabled Vuze Communication Procedure

A DECADE plugin can change the data path of BitTorrent download by using a "Redirect" message.

The detailed communication procedure is as following:

- When each client starts the download task ,it will try to connect the tracker to get peer list and then send "BT_Request" message to other peers in peer list.
- If the DECADE plugin is enabled, then it will intercept the incoming "BT_Request" message from other Vuze clients, and then reply with a "Redirect" message which includes DECADE server's address, authorization token and so on to the requester.
- When a DECADE plugin receives a "Redirect" message, it will connect to the DECADE server according to message context and send "Remote Get" message to the DECADE server to request the data, then wait for response.
- When a DECADE server receives a "Remote Get" message, it will check the server IP address in the message, Case 1: if the address equals to its own IP address, then it will send the data to the requester from its local disk/memory; Case 2: if the address is not equal to its own IP address, then it will send the "Remote Get" message to that address, to fetch the data, and then send the data to the requester. The data will be cached in the server locally for use by other requesters.
- When a DECADE plugin obtains the data, it will encapsulate the data into a standard "BT_Piece" message and send to Vuze client, then the client can write the data block into storage.

The detailed communication diagram is as follow:

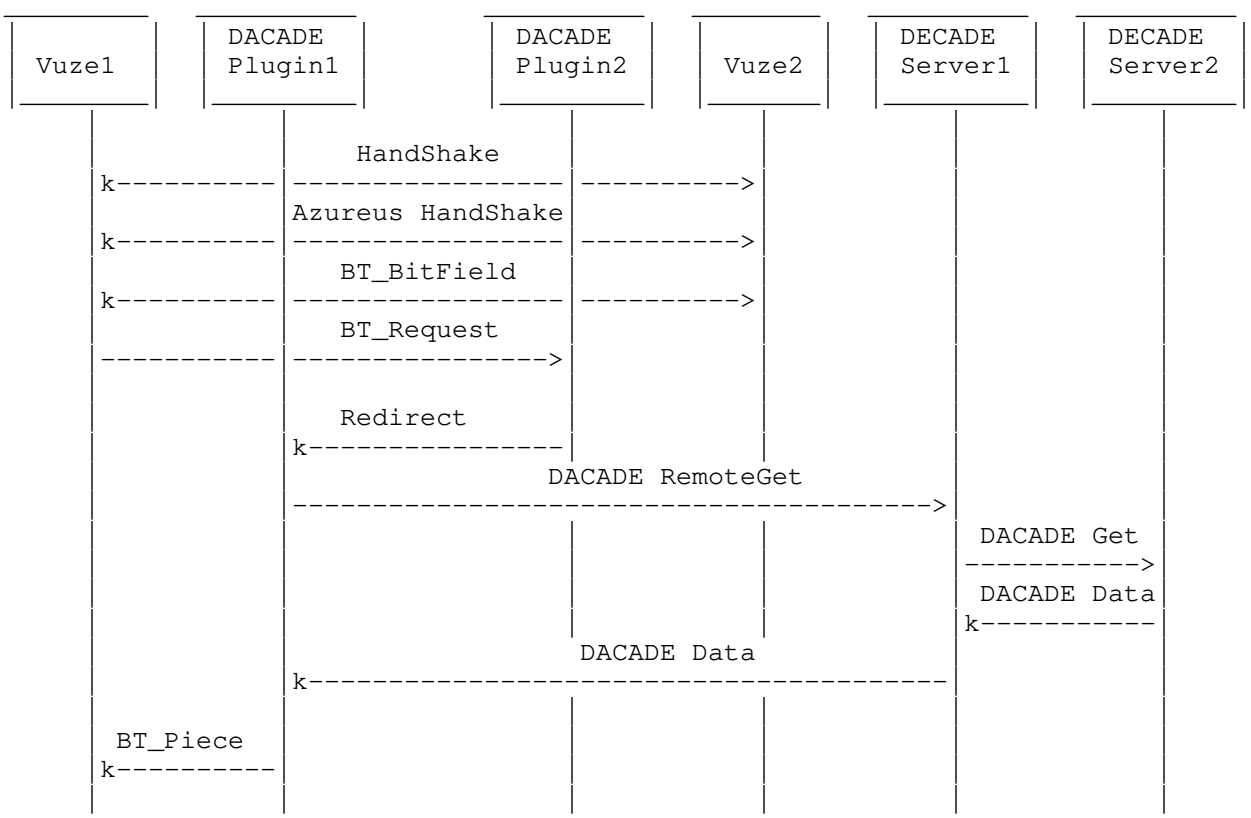


Figure 3

6. Test Environment and Settings

In order to demonstrate the performance of our DECADE implementation and DECADE-integrated P2P live streaming and file-sharing applications, we conduct some experimental tests in Amazon EC2 and PlanetLab. We perform a pair of comparative experiments: DECADE integrated P2P application v.s. native P2P application, in the same environment using the same settings.

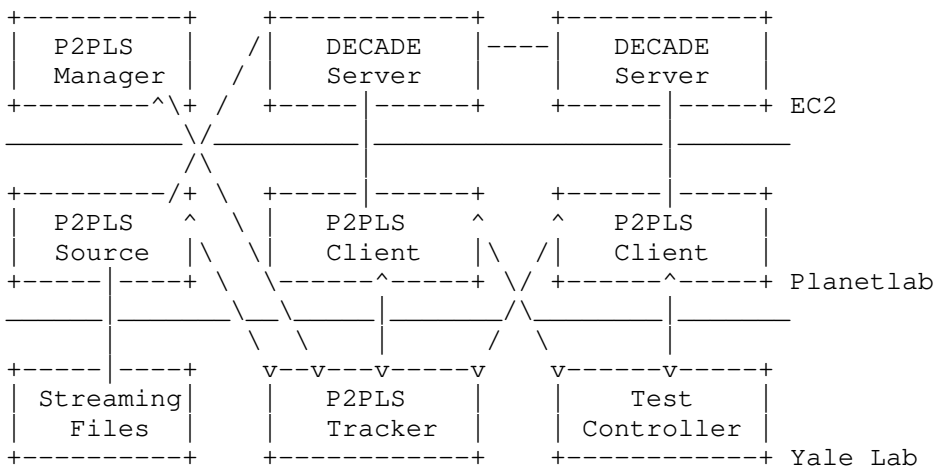
6.1. Test Settings

Our tests ran on a wide-spread area and diverse platforms, including a famous commercial cloud platform, Amazon EC2 and a well-known testbed, PlanetLab. The environment settings are as following:

- EC2 Regions: we setup DECADE servers in Amazon EC2 cloud, including all four regions around the world, US east, US west, Europe and Asia.
- PlanetLab: we run our P2P live streaming clients and P2P file-sharing clients (both DECADE integrated and native clients) on PlanetLab of a wild-spread area.
- Arrival pattern: we made all the clients join into the system within a short duration to simulate a flash crowd scenario.
- Total Bandwidth: for a fair comparison, we set the system's total supply bandwidth to be exact the same in both test.

6.2. Platforms and Components of P2PLS

In the tests, we have different functional components running in different platforms, including DECADE servers, P2P live streaming clients(DECADE integrated or Native), Tracker, Source server and Test Controller, as shown in Figure 4.



P2PLS represents to P2P LiveStreaming.

Figure 4

6.2.1. EC2 DECADE Server

DECADE Servers ran on Amazon EC2 small instances, with bandwidth constraint.

6.2.2. PlanetLab P2P LiveStreaming Client

Both DECADE integrated and Native P2P live streaming clients ran on planetlab which spreads in various locations around the world. The DECADE integrated P2P live streaming clients connect to the closest DECADE server according to its Geo-location distance to the servers. DECADE integrated P2P live streaming clients use their DECADE servers to upload to neighbors, instead of their own "last-mile" bandwidth.

6.2.3. Tracker

A native P2P live streaming tracker ran at Yale's laboratory and served both DECADE integrated clients and native clients during the test.

6.2.4. Source Server

A native P2P live streaming source server ran at Yale's laboratory and serve both DECADE integrated clients and native clients during the test. The capacity of source is equivalently constrain for both cases.

6.2.5. Test Controller

Test Controller is a manager to control all machines' behaviors in both EC2 and PlanetLab during the test.

6.3. Platforms and Components of Vuze

Test platforms includes Vuze client, tracker, DECADE Plugin, DECADE Servers and other supportive components such as HTTP Server, FTP Server and Remote Controller.

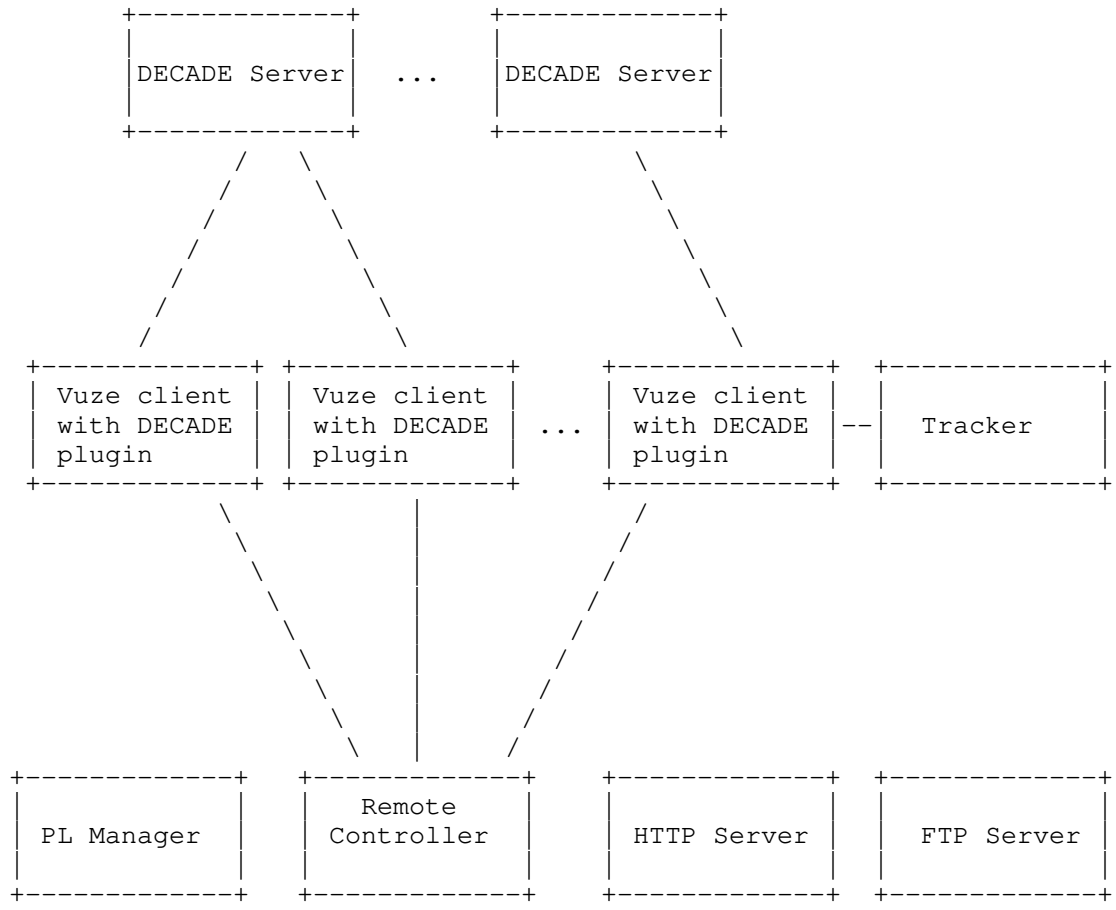


Figure 5

6.3.1. EC2 DECADE Server

DECADE Servers ran on Amazon EC2 small instances, with bandwidth constraint.

6.3.2. Vuze Client with DECADE Plugin

Vuze clients are divided into one seeding client and multiple leechers. Leechers run at PlanetLab, while the seeding client runs at the Window 2003 server. DECADE Plugin will be automatically loaded and run after Vuze client starts up.

6.3.3. Remote Controller

Remote controller can list all the Vuze clients in user interface and control them to download a specific BitTorrent file. It runs at the same Window 2003 server with the seeding client.

6.3.4. Tracker

Vuze client provides tracker capability, so we did not deploy our own tracker. Vuze embedded tracker is enabled when making a torrent file, the seeding client is also a tracker in this test.

6.3.5. HTTP Server

Torrent file will be put in the HTTP Server and the leechers will retrieve the torrent file from HTTP Server after receiving the download command from Remote Controller. We use Apache Tomcat which is an open source software as HTTP Server.

6.3.6. PL Manager

PL Manager (PlanetLab experiment Manager)is a tool developed by University of Washington, which presents a simple GUI to control PlanetLab nodes and perform common tasks such as:

- Selecting nodes for your slice.
- Choosing nodes for your experiment based on CoMon information about the nodes.
- Reliably deploying you experiment files.
- Executing commands / sets of commands on every node in parallel.
- Monitoring the progress of the experiment as a whole, as well as viewing console output from the nodes.

7. Performance Analysis

During the test, Both DECADE integrated P2P live streaming clients and DECADE integrated P2P file-sharing clients achieved impressively better performance than native clients.

7.1. Performance Metrics

7.1.1. P2P Live Streaming

To measure the performance of a P2P live streaming client, we employ mainly four metrics:

- Startup Delay: the duration from a peer joins the 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.

7.1.2. Vuze

For the performance comparison of Native Vuze and DECADE-Enabled Vuze, the same system bandwidth is provided through some parameter settings. In Native Vuze, the system bandwidth is defined as the amount of the uplink bandwidth from all the Vuze clients. The maximum upload bandwidth (Bu) is configured to every Vuze client before the test. Suppose the number of the Vuze clients is N, the system bandwidth is Bu * N. In the DECADE-Enabled Vuze case, the same bandwidth (Bu * N) is configured to the corresponding Ethernet port of DECADE Server.

7.2. Result and Analysis

7.2.1. P2P Live Streaming

- Startup Delay: In the test, DECADE integrated P2P live streaming clients startup around 35~40 seconds. some of them startup at about 10 seconds. Native P2P live streaming clients startup around 110~120 seconds. less than 20% of them startup within 100 seconds.

- Piece Missed Rate: In the test, both DECADE integrated P2P live streaming clients and native P2P live streaming clients achieved a good performance in pieces missed rate. Only about 0.02% of total pieces missed in both cases.

- Freeze Times: In the test, native P2P live streaming clients suffered from more freezing than DECADE Integrated P2P live streaming clients by 40%.

- Average Peer Uploading Rate: In the test, according to our settings, DECADE integrated P2P live streaming clients had no upload in their "last-mile" access network. While in the native P2P live streaming system, more than 70% of peers uploaded in a rate that is much more than streaming rate. In another word, DECADE can shift uploading traffic from clients' "last-mile" to in-network devices, which saves a lot of expensive bandwidth on access links..

7.2.2. Vuze

Performance advantage is shown according to the test result of experiment:

-

  There is no upload data traffic from Vuze clients in the DECADE-Enabled Vuze experiment, but in the Native Vuze experiment, the upload data traffic from Vuze clients is the same as download data traffic. Bandwidth resource is reduced in the last mile in the DECADE-Enabled Vuze experiment. The test result is illustrated in the following table. The download traffic and upload traffic include data traffic and signal traffic. For the upload traffic in the DECADE-enable Vuze, the data traffic is zero so the all traffic is signal overhead. Though we used the same torrent file in those two cases, the number of Vuze client was not the same because the nodes in the Planet-lab are not always stable. Therefore, the download traffic is a little different in two cases.

	Download traffic	Upload traffic
DECADE-Enabled Vuze	480MB	12MB
Native Vuze	430MB	430MB

Figure 6

o

Higher system resource efficiency in the DECADE-Enabled Vuze experiment, system resource efficiency is defined as the ratio of system download rate to the total system bandwidth. The test result is illustrated in the following figure.

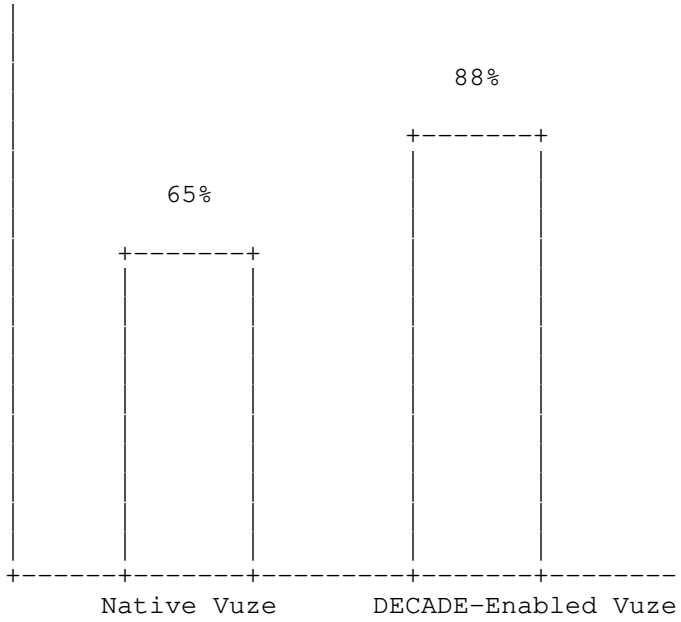


Figure 7

8. Security Considerations

This document does not contain any security considerations.

9. IANA Considerations

This document does not have any IANA considerations.

10. Normative References

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

Authors' Addresses

Lijiang Chen
Yale University

Email: lijiang.chen@yale.edu

Hongqiang Liu
Yale University

Email: hongqiang.liu@yale.edu

Zhigang Huang
HUAWEI Technologies

Email: hpanda@huawei.com

Xiaohui Chen
HUAWEI Technologies

Email: chen.xiaohui@huawei.com

Individual Submission
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

C. Dannewitz
University of Paderborn
T. Rautio
VTT Technical Research Centre of Finland
O. Strandberg
Nokia Siemens Networks
B. Ohlman
Ericsson

March 14, 2011

Secure naming structure and p2p application interaction
draft-dannewitz-ppsp-secure-naming-02

Abstract

Today, each application typically uses its own way to identify data. The lack of a common naming scheme prevents applications from benefiting from available copies of the same data distributed via different P2P and CDN systems. The main proposal presented in this draft is idea that there should be a secure and application independent way of naming information objects that are transported over the Internet. The draft defines a set of requirements for such a naming structure. It also presents a proposal for such a naming structure that could relevant for a number of work groups (existing and potential), e.g. PPSP, DECADE and CDNI. In addition, today's P2P naming schemes lack important security aspects that would allow the user to check the data integrity and build trust in data and data publishers. This is especially important in P2P applications as data is received from untrusted peers. Providing a generic naming scheme for P2P systems so that multiple P2P systems can use the same data regardless of data location and P2P system increases the efficiency and data availability of the overall data dissemination process. The secure naming scheme is providing self-certification such that the receiver can verify the data integrity, i.e., that the correct data has been received, without requiring a trusted third party. It also enables owner authentication to build up trust in (potentially anonymous) data publishers. The secure naming structure should be beneficial as potential design principle in defining the two protocols identified as objectives in the PPSP charter. This document enumerates a number of design considerations to impact the design and implementation of the tracker-peer signaling and peer-peer streaming signaling protocols.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",

"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Status of this Memo

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

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

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

This Internet-Draft will expire on September 15, 2011.

Copyright Notice

Table of Contents

1. Introduction

Today's dominating naming schemes in the Internet, i.e., IP addresses and URLs, are rather host-centric with respect to the fact that they are bound to a location. This kind of naming scheme is not optimal for many of the predominant users of todays Internet like P2P and CDN systems as they are based on an information-centric thinking, i.e., putting the information itself in focus. In these system the source of the information is secondary and can constantly change, e.g. new caches or P2P peers becomes available. It also common to retrieve information from more than one source at once.

For any type of caching solution (network based or P2P) and network based storage, e.g. DECADE, a common application independent naming scheme is essential to be able to identify cached copies of information/data objects.

Many applications, in particular P2P applications, use their own data model and protocol for keeping track of data and locations. This poses a challenge for use of the same information for several applications. A common naming scheme for information objects is important to enable interconnectivity between different application systems, such as P2P and CDN. To be able to build a common P2P infrastructure that can serve a multitude of applications there is a need for a common application independent naming scheme. With such a naming scheme different applications can use and refer to the same information/data objects.

It is possible to introduce false data into P2P systems, only detectable when the content is played out in the user application. The false data copies can be identified and sorted out if the P2P system can verify the reference used in the tracker protocol towards data received at the peer. One option to address this can be to secure the naming structure i.e. make the data reference be dependent on the data and related metadata.

An additional reason to introduce a common naming scheme for information objects is caching. When data are named in a host-centric way, as is done today, it is not always identify that copies of the same information object are available in multiple hosts. With location independent identifiers for information objects this becomes much easier.

This document enumerates and explains the rationale for why a common naming structure for information/data objects should be defined and used by a wide range of applications and network protocols. Examples of WGs (and potential WGs) where we think a new standard for naming of information/data objects should be valuable includes PPS, DECADE

and CDNI. For P2P systems the main advantage is probably in the definition of a protocol for signaling and control between trackers and peers (the PPSP "tracker protocol") but also a signaling and control protocol for communication among the peers (the PPSP "peer protocol") might have benefits from a common and secure naming scheme. In DECADE one key feature would be that different applications can easily share the same cache entries. It should also be valuable for cooperative caching, e.g. CDNI.

2. Naming requirements

In the following, we discuss the requirements that a common naming scheme has to fulfill.

To enable efficient, large scale data dissemination that can make use of any available data copy, identifiers (IDs) have to be location-independent. Thereby, identical data can be identified by the same ID independently of its storage location and improved data dissemination can then benefit from all available copies. This should be possible without compromising trust in data regardless of its network source.

Security in an information-centric network (including P2P, caching, and network-based solutions) needs to be implemented differently than in host-centric networks. In the latter, most security mechanisms are based on host authentication and then trusting the data that the host delivers. In e.g. a P2P system, host authentication cannot be relied upon, or one of the main advantages of a P2P system, i.e., benefiting from any available copy, is defeated. Host authentication of a random, untrusted host that happens to have a copy does not establish the needed trust. Instead, the security has to be directly attached to the data which can be done via the scheme used to name the data.

Therefore, self-certification is a main requirement for the naming scheme. Self-certification ensures the integrity of data and securely binds this data to its ID. More precisely, this property means that any unauthorized change of data with a given ID is detectable without requiring a third party for verification. Beforehand, secure retrieval of IDs (e.g., via search, embedded in a Web page as link, etc.) is required to ensure that the user has the right ID in the first place. Secure ID retrieval can be achieved by using recommendations, past experience, and specialized ID authentication services and mechanisms that are out of the scope of this discussion.

Another important requirement is name persistence, not only with

respect to storage location changes as discussed above, but also with respect to changes of owner and/or owner's organizational structure, and content changes producing a new version of the information. Information should always be identifiable with the same ID as long as it remains essentially equivalent. Spreading of persistent naming schemes like the Digital Object Identifier (DOI) [Paskin2010] also emphasizes the need for a persistent naming scheme. However, name persistence and self-certification are partly contradictory and achieving both simultaneously for dynamic content is not trivial.

From a user's perspective, persistent IDs ensure that links and bookmarks remain valid as long as the respective information exists somewhere in the network, reducing today's problem of "404 - file not found" errors triggered by renamed or moved content. From a content provider's perspective, name persistence simplifies data management as content can, e.g., be moved between folders and different servers as desired. Name persistence with respect to content changes makes it possible to identify different versions of the same information by the same consistent ID. If it is important to differentiate between multiple versions, a dedicated versioning mechanism is required, and version numbers may be included as a special part of the ID.

The requirement of building trust in an information-centric system combined with the desire for anonymous publication as well as accountability (at least for some content) can be translated into two related naming requirements. The first is owner authentication, where the owner is recognized as the same entity, which repeatedly acts as the object owner, but may remain anonymous. The second is owner identification, where the owner is also identified by a physically verifiable identifier, such as a personal name. This separation is important to allow for anonymous publication of content, e.g., to support free speech, while at the same time building up trust in a (potentially anonymous) owner.

In general, the naming scheme should be able to adapt to future needs. Therefore, the naming scheme should be extensible, i.e., it should be able to add new information (e.g., a chunk number for BitTorrent-like protocols) to the naming scheme. The need for such extensions is stressed by today's variety of naming schemes (e.g., DOI or PermaLink) added on top of the original Internet architecture that fulfill specialized needs which cannot be met by the common Internet naming schemes, i.e., IP addresses and URLs.

3. Basic Concepts for an Application-independent Naming Scheme

In this section, we introduce an exemplary naming scheme that illustrates a possible way to fulfill the requirements posed upon an

application-independent naming scheme for information-centric networks. The naming scheme integrates security deeply into the system architecture. Trust is based on the data's ID in combination with additional security metadata. Section 3.1 gives an overview of the naming scheme in general with details about the ID structure, and Section 3.2 describes the security metadata in more detail.

3.1. Overview

Building on an identifier/locator split, each data element, e.g., file, is given a unique ID with cryptographic properties. Together with the additional security metadata, the ID can be used to verify data integrity, owner authentication, and owner identification. The security metadata contains information needed for the security functions of the naming scheme, e.g., public keys, content hashes, certificates, and a data signature authenticating the content. In comparison with the security model in today's host-centric networks, this approach minimizes the need for trust in the infrastructure, especially in the host(s) providing the data.

In an information-centric network, multiple copies of the same data element typically exist at different locations. Thanks to the ID/ locator split and the application-independent naming scheme, those identical copies have the same ID and, hence, each application can benefit from all available copies.

Data elements are manipulated (e.g., generated, modified, registered, and retrieved) by physical entities such as nodes (clients or hosts), persons, and companies. Physical entities able of generating, i.e., creating or modifying data elements are called owners here. Several security properties of this naming scheme are based on the fact that each ID contains the hash of a public key that is part of a public/ secret key pair PK/SK. This PK/SK pair is conceptually bound to the data element itself and not directly to the owner as in other systems like DONA [Koponen]. If desired, the PK/SK pair can be bound to the owner only indirectly, via a certificate chain. This is important to note because it enables owner change while keeping persistent IDs. The key pair bound to the data is thus denoted as PK_D/SK_D.

Making the (hash of the) public key part of ID enables self-certification of dynamic content while keeping persistent IDs. Self-certification of static content can be achieved by simply including the hash of content in the ID, but this would obviously result in non-persistent IDs for dynamic content. For dynamic content, the public key in the ID can be used to securely bind the hash of content to the ID, by signing it with the corresponding secret key, while not making it part of ID.

The owner's PK as part of the ID inherently provides owner authentication. If the public key is bound to the owner's identity (i.e., to its real-world name) via a trusted third party certificate, this also allows owner identification. Without this additional certificate, the owner can remain anonymous.

To support the potentially diverse requirements of certain groups of applications and adapt to future changes, the naming scheme can enable flexibility and extensibility by supporting different name structures, differentiated via a Type field in the ID.

3.2. ID Structure

The naming scheme uses flat IDs to support self-certification and name persistence. In addition, flat IDs are advantageous when it comes to mobility and they can be allocated without an administrative authority by relying on statistical uniqueness in a large namespace, with the rare case of ID collisions being handled by the applications. Although IDs are not hierarchical, they have a specified basic ID structure. The ID structure given as ID = (Type field | A = hash(PK) | L) is described subsequently.

The Authenticator field A=Hash(PK_D) binds the ID to a public key PK_D. The hash function Hash is a cryptographic hash function, which is required to be one-way and collision-resistant. The hash function serves only to reduce the bit length of PK_D. PK_D is generated in accordance with a chosen public-key cryptosystem. The corresponding secret key SK_D should only be known to a legitimate owner. In consequence, an owner of the data is defined as any entity who (legitimately) knows SK_D.

The pair (A, L) has to be globally unique. Hence, the Label field L provides global uniqueness if PK_D is repeatedly used for different data.

To build a flexible and extensible naming scheme, e.g., to adapt the naming scheme to future changes, different types of IDs are supported by the naming scheme and differentiated via a mandatory and globally standardized Type field in each ID. For example, the Type field specifies the hash functions used to generate the ID. If a used hash function becomes insecure, the Type field can be exploited by the P2P system in order to automatically mark the IDs using this hash function as invalid.

3.3. Security Metadata Structure

The security metadata is extensible and contains all information required to perform the security functions embedded in the naming

scheme. The metadata (or selected parts of it) will be signed by SK_D corresponding to PK_D. This securely binds the metadata to the ID, i.e., to the Hash(PK_D) which is part of the ID. For example, the security metadata may include:

- specification of the hash function h and the algorithm DSAlg used for the digital signature
- complete PK_D (not only Hash(PK_D))
- specification of the parts of data that are self-certified, i.e., authenticated via the signature
- hash of the self-certified data
- signature of the self-certified data signed by SK_D
- all data required for owner authentication and identification

A detailed description and security analysis of this naming scheme and its security properties, especially self-certification, name persistence, owner authentication, and owner identification can be found in the GIS paper Secure Naming for a Network of Information [Dannewitz_10].

4. Examples of application use of secure naming structure

This section contains a number of examples how a secure naming system, as outlined in this draft, could be used by different types of applications.

4.1. Secure naming for P2P applications

From an P2P application perspective the main advantage of a secure naming structure for a P2P infrastructure is that multiple P2P applications can have common access to the same data elements. Another benefit of application-independent naming is that locally available and cached copies can easily be located. The secure naming also enables that data can be verified even if it is received from an untrusted host.

For example, when an application like BitTorrent [WWWbittorrent] uses self-certifying names, the user is guaranteed that the data received is actually the data that has been requested, without having to trust any servers in the network (e.g., the tracker) or the peers that provide the data.

This means that BitTorrent's validation of the data integrity can be improved significantly using the presented secure naming structure. Currently, a standard BitTorrent system has no means to verify the integrity of the torrent file and consequently of the data. The torrent file (see Figure 1) contains the SHA1 hashes of the content pieces (pieces in Figure 2). However, anyone can modify a torrent file to bind different content to this file. If the torrent file gets modified, the user has no means any more to verify the integrity of the data. Modification of the torrent affects only to info_hash value, which is SHA1 hash calculated from the torrent's info field (see figure). The info_hash is respectively used for torrent session identification in different software entities (e.g. in trackers). After changes in the torrent's info field, the torrent is referring to different torrent session that is carrying a forged content. Additionally if, the tracker allows insertion of several torrents with the same name - delivers forged data (consistent with the forged torrent file), a user could effectively be tricked into downloading forged content which would falsely be identified as being correct by the BitTorrent client. On the other hand, the torrent referring to a forged content can be also modified to point to, another, "convenient" tracker by modifying the announce field in the torrent, and the outcome would be the same from user perspective. I.e., in the current BitTorrent system, a user has no guarantee that the downloaded content actually matches the expected/correct content.

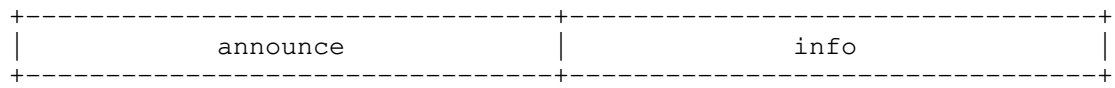


Figure 1: Basic structure of the BitTorrent torrent file

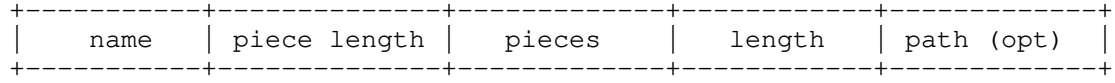


Figure 2: Structure of info field in torrent file

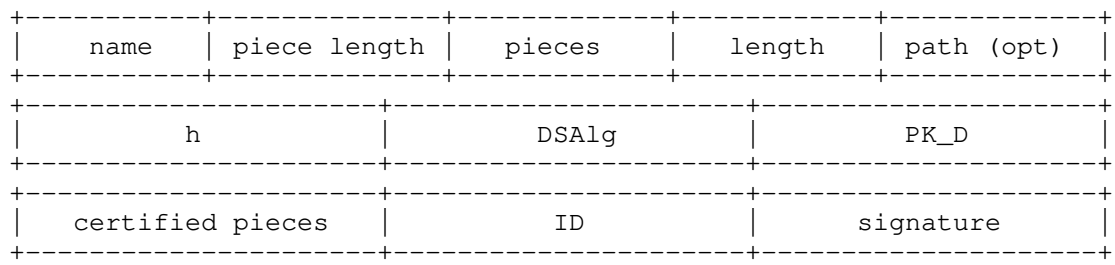


Figure 3: Structure of Secure naming enabled info field in torrent

The secure naming structure presented in this draft can provide a simple solution for this problem by securely binding the content of the torrent file to the name/ID of the torrent file. This can be done by extending the torrent file to include the above described security metadata information, as it is seen in Figure 3. In practice, during the torrent file creation, an object owner would store information about utilized algorithms (h - hash function and DSAlg - digital signature algorithm), the public key (PK_D), specification of signed data and ID into the torrent's info field, and will sign the combination of the secure metadata and the piece hash values (pieces in the torrent's info field) with the private key (SK_D). The generated signature will also be included in the extension part of the info field (signature).

After the content of the extended torrent is created, the respective torrent file ID would be generated according to the rules described in Section 3. As defined in that section, ID contains three different fields, namely Type, A and L. In the case of BitTorrent, Type field would carry on information about used hash function to generate field A from PK_D, and also structure of the field L. If, for example, L has name and version of the distributed file, Type field should tell that by including strings "Name" and "Version" in it. The next one, field A, includes hash values of the used PK_D (method defined in Type). And finally the proposed BitTorrents ID field L, can take in name and version of the distributed file. According to the description and by using separators - (within one field) and _ (between fields) the torrent file name could look, for example, like: HashMethod-Name-Version_HashofPK_Filename-Fileversion.torrent.

Consequently, whenever a user knows the ID of the content/torrent file and retrieves the torrent file, she/he can now open the torrent with the secure naming supported BitTorrent client. The client verifies the integrity of the torrent file by comparing PK_D in secure metadata and field A in the ID, in addition, conformance of ID in the torrent name and ID in the metadata is verified. With respect

to the secure metadata the signature and actual data is compared also. Once these three are verified, the client can download the data pieces, and can use the BitTorrent's included (and now secured) hash(es) to verify the integrity of the received data. As a result, the user can be sure that the correct content was retrieved.

4.2. Secure naming use in DECADE

DECADE WG is specifying requirements for a protocol concerning accessing data in a network storage and resource control of said data. A key aspect in accessing data in a network storage is the way the data is referenced. This naming draft has outlined a naming structure that can be utilized to enhance the reference to include additional features. The secure naming structure tries to fulfill several design targets and requirements, however not all are necessarily the priority requirements for the DECADE scope.

The DECADE storage is used by individual and uncoordinated entities, thus the naming of the data must be collision free. Also when a user accesses data the name should point to the correct data. With no entity to keep track of used names for data, one potential approach is to use large enough identifier designed with statistically collision free random properties. One obvious identifier alternative is to base it on the hash of the content.

The basic requirement for naming in DECADE is that the data identifier is tied to the hash of the content and that it is taken from a large enough flat namespace. In this way, wherever the same data is stored, the same name identifier can be used. Someone accessing the data can verify that the content is correct based on relationship between data and identifier. Other requirements can be included to further enhance the meaning and capability of the data reference identifier. Additional naming requirements could be:

- self-certified name for verifying content owner (owner of Pk/Sk keys), the self-certification can be used for building trust about data publisher
- solution for persistent identifier names for dynamic (changing) data
- potentially way to identify content owner, this typically requires trusted third party certifier.

This draft specifies several requirements that would be useful for the DECADE protocol, the main requirement is hashing of data into the identifier name. Depending of data use (like enhanced security properties) other secondary requirements will be beneficial for

additional functionality.

4.3. Secure naming for CDNs

The use of common naming within a CDN is not the challenge, it is the common naming between end users and the CDN or between CDNs that can be feasible. A common naming enables use of numerous caching points and CDNs as the same data can be referenced in the same way. The same data can depending of popularity be available in multiple location like caches and CDNs. The population of the resources (caches and CDNs) can be efficient if a common naming of data is used. The interaction between CDNs to 'negotiate' data population of caching resources would benefit from a common data reference model. The security features of the naming scheme also helps the CDN provider trust the data it is accessing and providing. The CDN interaction is potentially in the scope of the CDNI WG BOF.

5. Conclusion

The main proposal presented in this draft is idea that there should be a secure and application independent way of naming information objects that are transported over the Internet. The draft defines a set of requirements for such a naming structure. It also presents a proposal for such a naming structure that could relevant for a number of work groups (existing and potential), e.g. PPSP, DECADE and CDNI.

Specifically for the PPSP WG the secure naming structure is proposed for consideration as common reference ID structure. For any P2P streaming application to have fair and multitude of data access, it is essential to have a common naming structure that is suitable for many different needs. The common naming is probably best displayed in the tracker protocol case but potential benefit in the actual streaming protocol case has to still be identified. The secure binding of reference ID to the actual content is manifested in the end user peer possibility to check correct data reception in regard to the used ID.

The naming structure has been implemented in the 4WARD project prototypes and has been released as open source (www.netinf.org). The naming structure is also available through a public NetInf registration service at www.netinf.org. Three NetInf-enabled applications have also been published, the InFox (Firefox plugin), InBird (Thunderbird plugin), and a NetInf Information Object Management Tool, all available at the www.netinf.org site.

Continued work on defining a common naming structure for information objects is carried out in the SAIL project. More information is

available at the www.sail-project.eu site.

6. IANA Considerations

This document has no requests to IANA.

7. Security Considerations

There are considerations about what private/public key and hash algorithms to utilize when designing the naming structure in a secure way.

8. Acknowledgements

We would like to thank all the persons participating in the Network of Information work packages in the EU FP7 projects 4WARD and SAIL and the Finnish ICT SHOK Future Internet 2 project for contributions and feedback to this document.

9. Informative References

[Dannewitz_10] Dannewitz, C., Golic, J., Ohlman, B., and B. Ahlgren, "Secure Naming for a Network of Information", 13th IEEE Global Internet Symposium , 2010.

[Koponen] Koponen, T., Chawla, M., Chun, B., Ermolinskiy, A., Kim, K., Shenker, S., and I. Stoica, "A Data-Oriented (and beyond) Network Architecture", Proc. ACM SIGCOMM , 2007.

[Paskin2010] Paskin, N., "Digital Object Identifier ({DOI}?) System", Encyclopedia of Library and Information Sciences , 2010.

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

[WWWbittorrent] Cohen, B., "The BitTorrent Protocol Specification", http://www.bittorrent.org/beps/bep_0003.html , 2008.

Authors' Addresses

Christian Dannewitz
University of Paderborn
Paderborn
Germany

Email: cdannewitz@upb.de

Teemu Rautio
VTT Technical Research Centre of Finland
Oulu
Finland

Email: teemu.rautio@vtt.fi

Ove Strandberg
Nokia Siemens Networks
Espoo
Finland

Email: ove.strandberg@nsn.com

Borje Ohlman
Ericsson
Stockholm
Sweden

Email: Borje.Ohlman@ericsson.com

DECADE
Internet-Draft
Intended status: Standards Track
Expires: September 8, 2011

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

March 7, 2011

DECADE Architecture
draft-ietf-decade-arch-00

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. One technique to improve the network efficiency of P2P applications is to introduce storage capabilities within the networks. The DECADE Working Group has been formed with the goal of developing an architecture to provide this capability. This document presents an architecture, discusses the underlying principles, and identifies core components and protocols supporting the architecture.

This Internet-Draft will expire on September 8, 2011.

Table of Contents

1. Introduction

Peer-to-peer (P2P) applications have become widely used on the Internet today to distribute contents, and they contribute a large portion of the traffic in many networks. The DECADE Working Group has been formed with the goal of developing an architecture to introduce in-network storage to be used by such applications, to achieve more efficient content distribution. Specifically, in many subscriber networks, it is typically more 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 and CMTSs. On the other hand, it can be cheaper to upgrade the core infrastructure, which involves fewer components that are shared by many subscribers. See [I-D.ietf-decade-problem-statement] for a more complete discussion of the problem domain and general discussions of the capabilities to be provided by DECADE.

This document presents a potential architecture of 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. In particular, content distribution applications that may split data into smaller pieces for distribution may be able to utilize DECADE.

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

The DECADE protocol will leverage existing transport and application layer protocols and will be designed to work with a small set of alternative IETF protocols.

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

This document will be updated to track the progress of the DECADE survey [I-D.ietf-decade-survey] and requirements [I-D.gu-decade-reqs] drafts.

2. Entities

2.1. DECADE Storage Servers

DECADE storage servers are operated by DECADE storage providers and provide the DECADE functionality as specified in this document, including mechanisms to store, retrieve and manage data. A storage provider may typically operate multiple storage servers.

2.2. DECADE Storage Provider

A DECADE in-network storage provider deploys and/or manages DECADE servers within a network. A storage provider may also own or manage the network in which the DECADE servers are deployed.

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.

2.3. DECADE Content Providers

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

2.4. DECADE Content Consumers

DECADE content consumers access storage servers (by way of a DECADE client) to download data that has previously been stored by a content provider. A content consumer can access one or more storage servers. A content consumer may be a single process or a distributed application (e.g., in a P2P scenario). An instance of a distributed application, such as a P2P application, may both provide content to and consume content from DECADE storage servers.

2.5. Content Distribution Application

A content distribution application is a distributed application designed for dissemination of possibly-large data to multiple consumers. Content Distribution Applications typically divide content into smaller immutable blocks for dissemination.

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

2.6. 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.

An Application End-Point need not be an active member of a "swarm" to interact with the DECADE storage system. That is, an End-Point may interact with the DECADE storage servers as an offline activity.

3. Architectural Principles

We identify the following key principles.

3.1. Decoupled Control and Data Planes

The DECADE infrastructure is intended to support multiple content distribution applications. A complete content distribution application implements a set of control functions including content search, indexing and collection, access control, ad insertion, replication, request routing, and QoS scheduling. Different content distribution applications can have unique considerations designing the control and signaling functions. For example, a major competitive advantage of many successful P2P systems is their substantial expertise in achieving highly efficient utilization of peer and infrastructural resources. For instance, many live P2P systems have their specific algorithms in constructing topologies to achieve low-latency, high-bandwidth streaming. They continue to fine-tune such algorithms. In other words, in-network storage should export basic mechanisms and allow as much flexibility as possible to the control planes to implement specific policies. This conforms to the end-to-end systems principle and allows innovation and satisfaction of specific business goals.

Specifically, in the DECADE architecture, the control plane focuses on the application-specific, complex, and/or processing intensive functions while the data plane provides storage and data transport functions.

- Control plane: Signals details of where the data is to be downloaded from. The control signals may also include the time, quality of service, and receivers of the download. It also provides higher layer meta-data management functions such as defining the sequence of data blocks forming a higher layer content object. These are behaviors designed and implemented by the Application. By Application, we mean the broad sense that

includes other control plane protocols.

- Data plane: Stores and transfers data as instructed by the Application's Control Plane.

Decoupling control plane and data plane is not new. For example, OpenFlow is an implementation of this principle for Internet routing, where the computation of the forwarding table and the application of the forwarding table are separated. Google File System 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). NFS4 also implements this principle.

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

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

3.2. Immutable Data Objects

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

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

Common applications whose content matches this model include P2P streaming (live and video-on-demand) and P2P file-sharing content. However, other additional types of applications may match this model.

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

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

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

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

3.3. Data Object Identifiers

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

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

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

3.4. Explicit Control

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

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

3.5. Resource and Data Access Control through User Delegation

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

3.5.1. Resource Allocation

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

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

3.5.2. User Delegations

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

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

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

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

4. System Components

The primary focus of the current version of this document is on the architectural principles. The detailed system components will be discussed in the next document revision.

This section presents an overview of the components in the DECADE architecture.

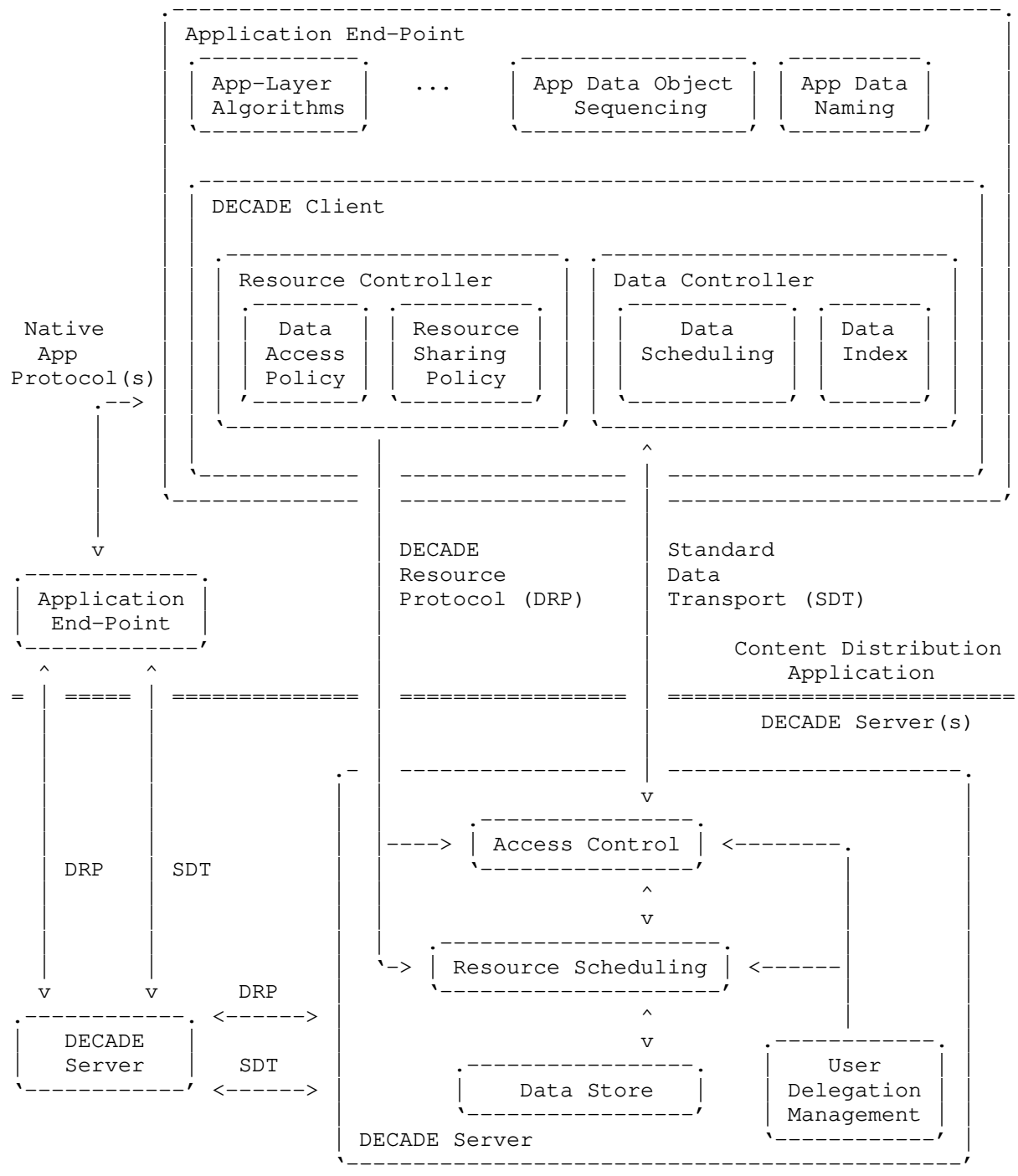

Application End-Point
App-Layer Algorithms
...
App Data Object Sequencing
App Data Naming
DECADE Client
Resource Controller
Data Access Policy
Resource Sharing Policy
Data Controller
Data Scheduling
Data Index
Native App Protocol(s)
Application End-Point
DECADE Resource Protocol (DRP)
Standard Data Transport (SDT)
Content Distribution Application
DECADE Server(s)
Access Control
Resource Scheduling
Data Store
User Delegation Management
DRP
SDT
DECADE Server
DECADE Server
DRP
SDT

Figure 1: DECADE Architecture Components

A component diagram of the DECADE architecture is displayed in Figure 1. The diagram illustrates the major components of a Content Distribution Application related to DECADE, as well as the functional components of a DECADE Server.

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

4.1. Content Distribution Application

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

4.1.1. Data Sequencing and Naming

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

In assembling or producing the data objects, an important consideration is the naming of these objects. We call the component responsible for assigning and interpreting application-layer names the Application Data Naming component. The specific implementation is entirely decided by the application.

4.1.2. Native Protocols

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

4.1.3. DECADE Client

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

4.1.3.1. Resource Controller

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

4.1.3.2. Data Controller

DECADE is designed to decouple the control and the data transport of applications. Data transport between applications and DECADE servers uses standard data transport protocols. It may need to schedule the data being transferred according to network conditions, available DECADE Servers, and/or available DECADE Server resources. An index indicates data available at remote DECADE servers. The index (or a subset of it) may be advertised to other Application End-Points.

4.2. DECADE Server

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

4.2.1. Access Control

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

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

4.2.2. Resource Scheduling

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

Access control and resource control are separated in DECADE server. It is possible that an Application End-Point provides only access to its data without any resources. In order to access this data, another Application End-Point may use the granted access along with its own available resources to store or retrieve data from a DECADE Server.

4.2.3. Data Store

Data from applications may be stored into disks. Data can be deleted from disks either explicitly or automatically (e.g., after a TTL). It may be possible to perform optimizations in certain cases, such as avoiding writing temporary data (e.g., live streaming) to a disk.

4.3. Protocols

The DECADE Architecture uses two protocols. First, the DECADE Resource Protocol is responsible for communicating access control and resource scheduling policies to the DECADE Server. Second, standard data transport protocols (e.g., WebDAV or NFS) are used to transfer data objects to and from a DECADE Server. The DECADE architecture will specify a small number of Standard Data Transport instances.

Decoupling the protocols in this way allows DECADE to both directly utilize existing standard data transports and to evolve independently.

It is also important to note that the two protocols do not need to be separate on the wire. For example, the DECADE Resource Protocol messages may be piggybacked within the extension fields provided by certain data transport protocols. However, this document considers them as two separate, logical functional components for clarity.

4.3.1. DECADE Resource Protocol

The DECADE Resource Protocol is responsible for communicating both access control and resource sharing policies to DECADE Servers used for data transport.

The DECADE architecture specification will provide exactly one DECADE Resource Protocol.

4.3.2. Standard Data Transports

Existing data transport protocols are used to read and write data from a DECADE Server. Protocols under consideration are WebDAV and NFS.

4.4. DECADE Data Sequencing and Naming

We have discussed above that an Application may have its own behavior for both sequencing and naming data objects. In order to provide a simple and generic interface, the DECADE Server is only responsible for storing and retrieving individual data objects.

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

Multiple applications may make use of a DECADE infrastructure, and each Application may employ its own naming scheme. To remain independent of particular applications, DECADE uses a simple, common naming scheme that supports unique naming of individual data objects. This is achieved by deriving object names from hashes of the object content. This scheme is made possible by the fact that DECADE data objects are immutable. Details of the naming scheme (complete syntax, hash algorithms etc.) will be defined in a future version of this document.

By naming data objects directly as the content hash, DECADE names satisfy important objectives:

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

A particular advantage of using the content hash is that it is straightforward for as server or client to validate a data object before storing or transmitting it. While these capabilities could be achieved by supplying the content hash in both read and write requests as metadata, using the content hash as the name satisfies the objectives and is straightforward.

Another advantage of this scheme is that a DECADE client knows the name of a data object before it is completely stored at the DECADE

server. This allows for particular optimizations, such as advertising data object while the data object is being stored, removing store-and-forward delays. For example, a DECADE client A may simultaneously begin storing an object to a DECADE server, and advertise that the object is available to DECADE client B. If it is supported by the DECADE server, client B may begin downloading the object before A is finished storing the object.

4.5. 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 [I-D.ietf-decade-survey] map into the DECADE architecture.

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

4.5.1. Data Access Interface

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

4.5.2. Data Management Operations

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

4.5.3. Data Search Capability

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

4.5.4. Access Control Authorization

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

4.5.5. Resource Control Interface

Users can manage the resources (e.g. bandwidth) on the DECADE server that can be used by other Application End-Points. Resource Sharing

Policies are generated by a Content Distribution Application and provided to the DECADE Client's Resource Controller. The DECADE Server is responsible for implementing the resource sharing policies.

4.5.6. Discovery Mechanism

This is outside the scope of the DECADE architecture. However, it is expected that DNS or some other well known protocol will be used for the users to discover the DECADE servers.

4.5.7. Storage Mode

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

5. DECADE Protocols

This section specifies the DECADE Resource Protocol (DRP) and the Standard Data Transport (SDT) in terms of abstract protocol interactions that are intended to mapped to specific protocols such as HTTP. It is possible that a single specific protocol provides both, DRP and SDT functionality.

The DRP is the protocol used by a DECADE client to configure the resources and authorization used to satisfy requests (reading, writing, and management operations concerning DECADE objects) at a DECADE server. The SDT is used to send the operations to the DECADE server. Necessary DRP metadata is supplied using mechanisms in the SDT that are provided for extensibility (e.g., additional request parameters). A detailed architectural description of the DRP and SDT is still a work in progress. Important aspects, including details of authorization, will be added in a future version of this document.

5.1. DECADE Resource Protocol (DRP)

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

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

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

storage resources: DECADE servers may limited storage resources.

Note: this list of resources will be extended in a future version of this document.

5.2. Standard Data Transport (SDT)

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

5.2.1. Writing/Uploading Objects

For writing objects, a client uploads an object to a DECADE server. The object on the server will be named (associated to an identifier), and this name can be used to access (download) the object later, e.g., the client can pass the name as a reference to other client that can then refer to the object.

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

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

The application that originates the objects MUST generate DECADE object names according to the naming specification in Section 4.4. The naming scheme provides that the name is unique. DECADE clients (as parts of application entities) upload a named object to a server, and a DECADE server MUST not change the name. It MUST be possible for downloading clients, to access the object using the original name. A DECADE server MAY verify the integrity and other security properties of uploaded objects.

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

Method PUT:

Parameters:

NAME: The naming of the object according to Section 4.4

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

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

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

OK: The object has been uploaded successfully and has replaced an existing object with the same name.

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

ERRORs: possible error codes later will be specified in a later version of this document

5.2.2. Downloading Objects

A DECADE client can request named objects from a DECADE server. In the following, we provide an abstract specification of the download operation that we name 'GET METHOD'. See Section 4.4 for an example how this could be mapped to HTTP.

Method GET:

Parameters:

NAME: The naming of the object according to Section 4.4.

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

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

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

ERRORs:

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

Other Errors: TBD in a future version of this document

6. Server-to-Server Protocols

An important feature of DECADE is the capability for one DECADE server to directly download data objects from another DECADE server. This capability allows Applications to directly replicate data objects between servers without requiring end-hosts to use uplink capacity to upload data objects to a different DECADE server. Similar to other operations in DRP and SDT, replicating data objects between DECADE servers is an explicit operation.

To support this functionality, DECADE re-uses the already-specified protocols to support operations directly between servers. DECADE servers are not assumed to trust each other nor are configured to do so. All data operations are performed on behalf of DECADE clients via explicit instruction, so additional capabilities are needed in the DECADE client-server protocols DECADE clients must be able to indicate to a DECADE server the following additional parameters:

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

In this way, a DECADE server is also a DECADE client, and requests may instantiate requests via that client. The operations are performed as if the original requestor had its own DECADE client co-located with the DECADE server. It is this mode of operation that provides substantial savings in uplink capacity.

6.1. Operational Overview

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

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

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

TOKEN: Credentials to be used at the remote server.

These parameters are used by the DECADE server to instantiate a request to the specified remote server. It is assumed that the data object referred to at the remote server is the same as the original request. It is also assumed that the operation performed at the remote server is the same as the operation in the original request. Though explicitly supplying these may provide additional freedom, it is not clear what benefit they might provide.

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

In the case of a GET operation, the DECADE server is to retrieve the data object from the remote server using the specified credentials (via a GET request to the remote server), and then return the object to the client. In the case of a PUT operation, the DECADE server is to store the object from the client, and then store the object to the remote server using the specified credentials (via a PUT request to the remote server).

6.2. Potential Optimizations

As a suggestion to the protocol and eventual implementations, we would like to point out particular optimizations that could be made when making use of the server-to-server protocol.

6.2.1. Pipelining to Avoid Store-and-Forward Delays

A DECADE server may choose to not fully store an object before beginning to serve it. For example, in the case of a GET request, a DECADE server may begin to receive a data object from a remote server, and immediately begin returning it to the DECADE client. This pipelining mode avoids store-and-forward delays, which could be substantial for large objects. A similar behavior could be used for PUT.

7. Security Considerations

This document currently does not contain any security considerations beyond those mentioned in [I-D.ietf-decade-problem-statement].

8. IANA Considerations

This document does not have any IANA considerations.

9. Informative References

[I-D.ietf-decade-problem-statement]
Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", draft-ietf-decade-problem-statement-02 (work in progress), January 2011.

[I-D.ietf-decade-survey]
Alimi, R., Rahman, A., and Y. Yang, "A Survey of In-network Storage Systems", draft-ietf-decade-survey-04 (work in progress), March 2011.

[I-D.gu-decade-reqs]
Yingjie, G., Bryan, D., Yang, Y., and R. Alimi, "DECADE Requirements", draft-gu-decade-reqs-05 (work in progress), July 2010.

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

In this section we illustrate how the abstract protocol interactions specified in Section 5 for DECADE DRP and SDT can be fulfilled by existing protocols such as HTTP and WEBDAV. (This version of the document considers HTTP only, a future version will provide a possible WEBDAV-based illustration as well.)

A.1. HTTP

HTTP is a key protocol for the Internet and specifically the World Wide Web. HTTP is a request-response protocol. A typical transaction is when a client (e.g. web browser) requests content (resources) from a web server. Other examples are when the client puts or deletes content from the server.

A.1.1. HTTP Support for DECADE Resource Protocol Primitives

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

A.1.1.1. Access Control Primitives

Access control requires mechanisms for defining the access policies for the server, and then checking the authorization of a user before it stores or retrieves content. HTTP supports access control via "HTTP Secure" (HTTPS). HTTPS is a combination of HTTP with SSL/TLS. The main purpose of HTTPS is to authenticate the server and encrypt all traffic between the client and the server. HTTPS does not support authentication of the client.

A.1.1.2. Communication Resource Controls Primitives

Communications resources include bandwidth (upload/download) and number of simultaneous connected clients (connections). HTTP supports communication resource control through "persistent" HTTP connections. Persistent HTTP connections allows a client to keep open the underlying TCP connection to the server to allow streaming and pipelining (multiple simultaneous requests). HTTP does not support a mechanism to allow limiting the communciation resources to a client.

A.1.1.3. Storage Resource Control Primitives

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

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

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

A.1.2.1. Writing Primitives

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

existing resource or to create a new resource.

A.1.2.2. Downloading Primitives

Downloading involves fetching of an object from the server. HTTP supports downloading through the GET method. GET fetches a specific resource as identified by the URL.

A.1.2.3. Other Methods

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

Authors' Addresses

Richard Alimi
Google

Email: ralimi@google.com

Y. Richard Yang
Yale University

Email: yry@cs.yale.edu

Akbar Rahman
InterDigital Communications, LLC

Email: akbar.rahman@interdigital.com

Dirk Kutscher
NEC

Email: dirk.kutscher@neclab.eu

Hongqiang Liu
Yale University

Email: hongqiang.liu@yale.edu

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

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

February 7, 2013

DECADE Protocol
draft-ietf-decade-arch-10

Abstract

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

Requirements Language

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

Status of this Memo

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

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

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

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

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

Table of Contents

1. Introduction

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

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

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

2. Terminology

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

3. Protocol Flow

3.1. Overview

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

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

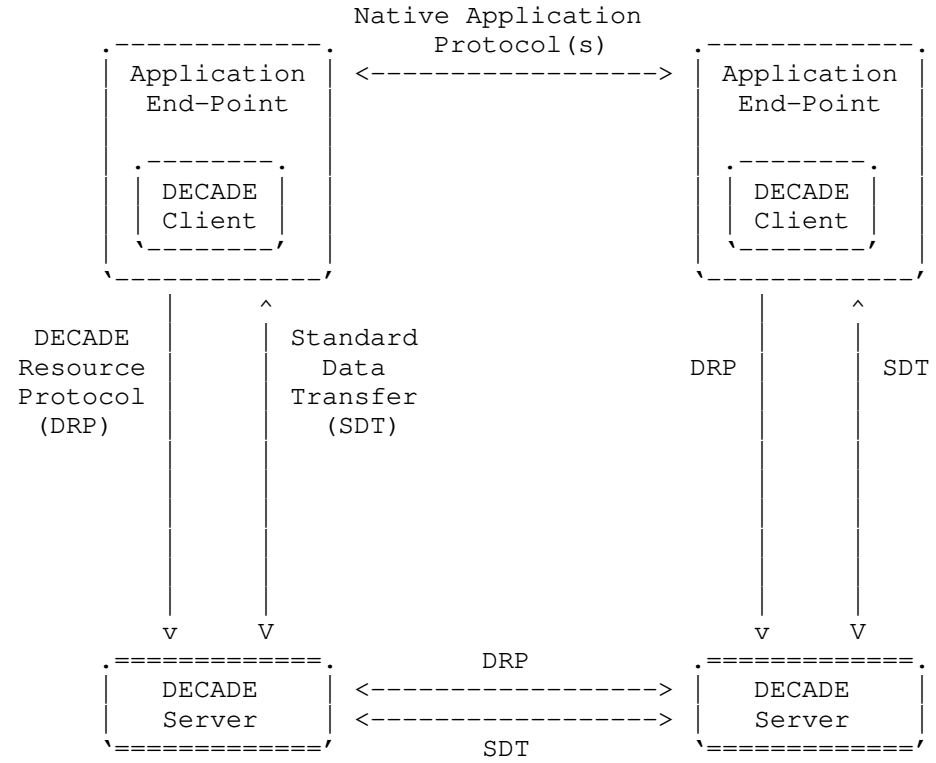


Figure 1: Generic Protocol Flow

3.2. An Example

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

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

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

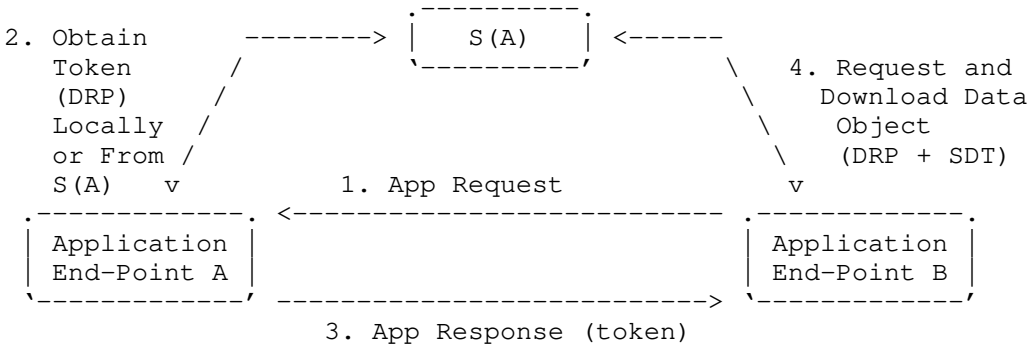


Figure 2: Download from Storage Server

4. Architectural Principles

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

4.1. Decoupled Control/Metadata and Data Planes

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

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

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

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

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

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

4.2. Immutable Data Objects

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

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

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

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

on minimum and maximum sizes by compliant implementations.

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

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

4.3. Data Objects With Identifiers

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

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

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

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

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

A DECADE naming scheme follows the following general requirements:

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

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

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

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

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

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

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

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

The naming scheme MAY additionally provide the following name elements:

- Application or publisher information.

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

4.4. Explicit Control

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

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

4.5. Resource and Data Access Control through Delegation

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

4.5.1. Resource Allocation

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

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

4.5.2. User Delegations

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

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

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

5. System Components

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

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

5.1. Content Distribution Application

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

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

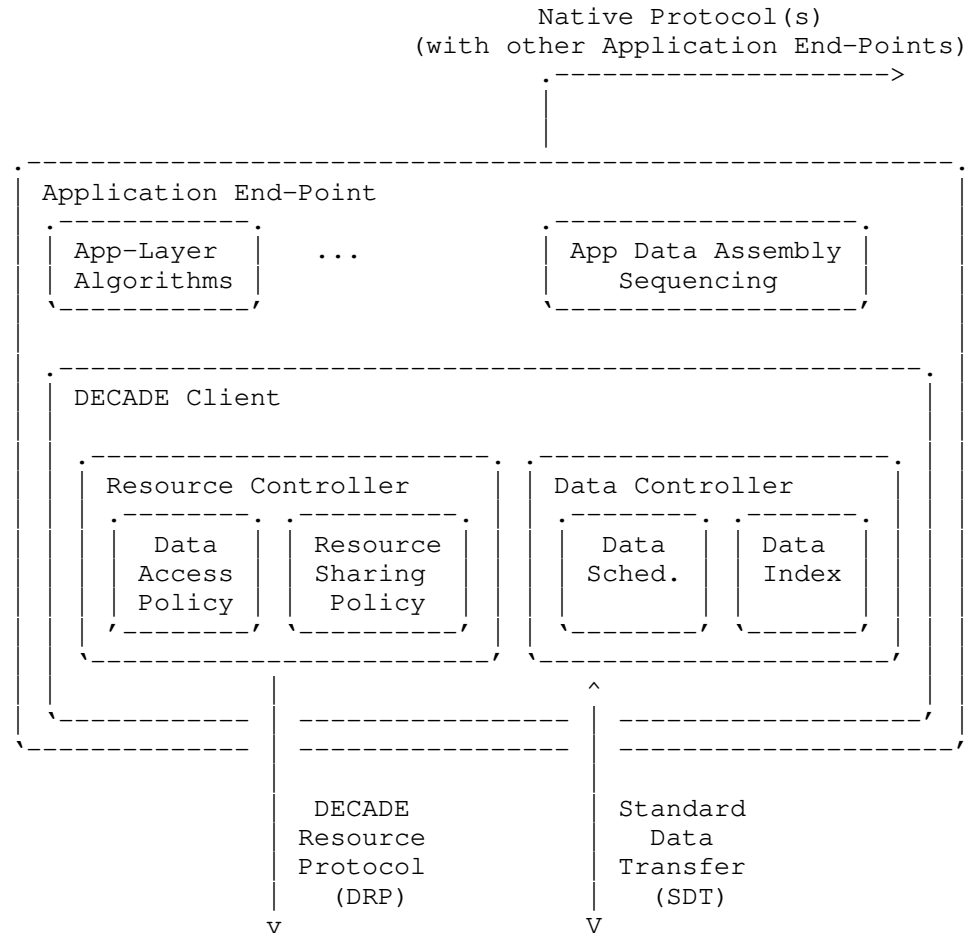


Figure 3: Application Components

5.1.1. Data Assembly

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

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

5.1.2. Native Application Protocols

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

5.1.3. DECADE Client

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

5.1.3.1. Resource Controller

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

5.1.3.2. Data Controller

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

5.2. DECADE Server

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

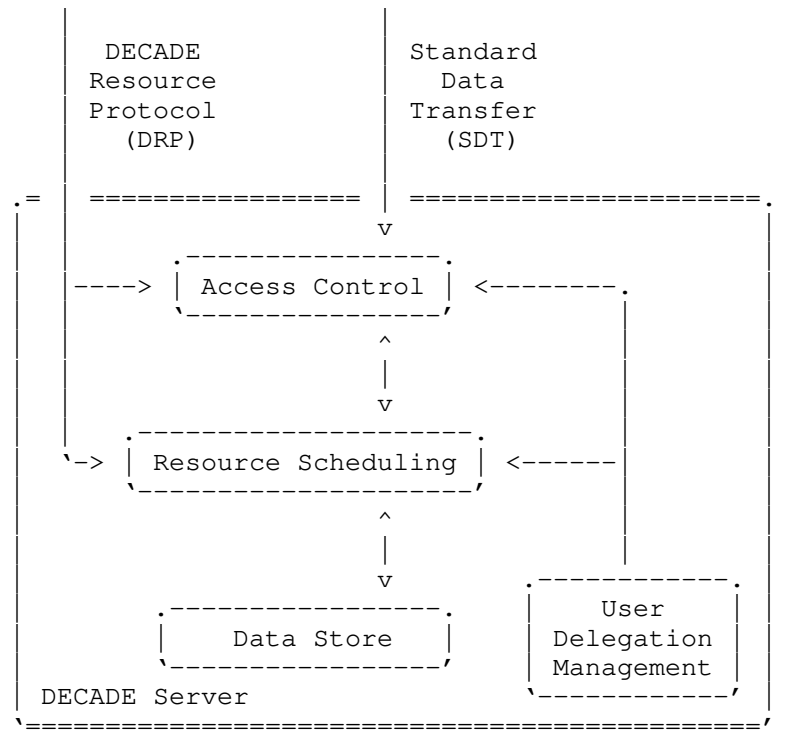


Figure 4: DECADE Server Components

5.2.1. Access Control

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

5.2.2. Resource Scheduling

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

5.2.3. Data Store

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

after a TTL expiration).

5.3. Data Sequencing and Naming

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

5.3.1. Application Usage Example

To illustrate these properties, this section presents multiple examples.

5.3.1.1. Application with Fixed-Size Chunks

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

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

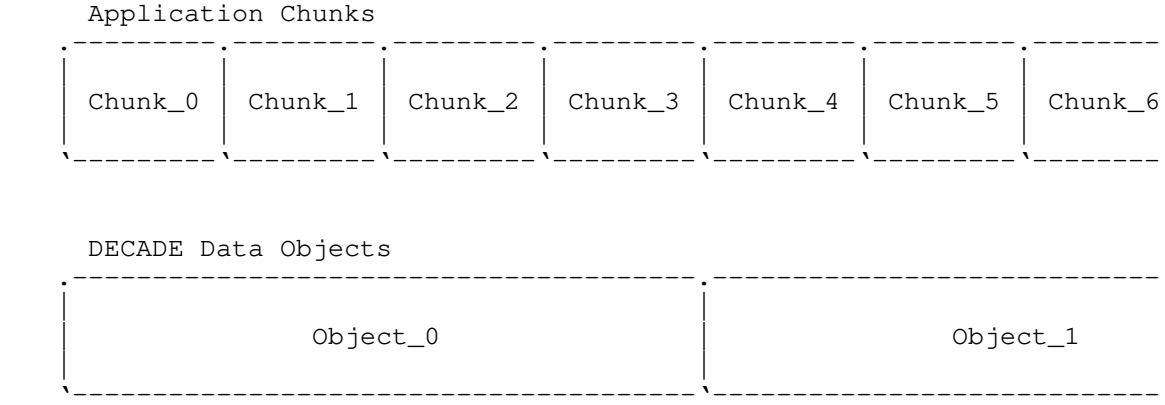


Figure 5: Mapping Application Chunks to DECADE Data Objects

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

data objects have globally unique names.

5.3.1.2. Application with Continuous Streaming Data

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

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

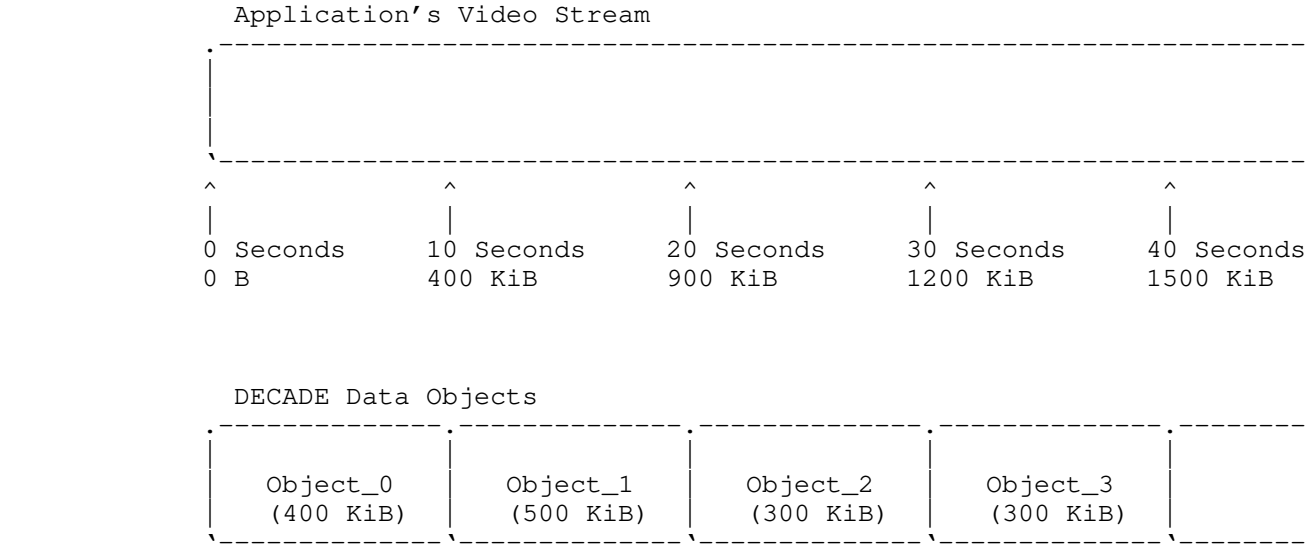


Figure 6: Mapping a Continuous Data Stream to DECADE Data Objects

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

5.4. Token-based Authorization and Resource Control

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

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

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

- 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 based on the delegation concept as described in Section 4.5. The following type of access information elements MAY be requested:

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

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

6.2.4. Data Object Attributes

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

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

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

Data Object size: In bytes;

Media type Labelling of type as per [RFC4288];

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

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

6.3. Standard Data Transfer (SDT) Protocol

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

6.3.1. Writing/Uploading Objects

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

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

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

6.3.2. Downloading Data Objects

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

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

message.

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

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

6.4. Server-to-Server Protocols

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

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

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

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

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

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

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

7. Security Considerations

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

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

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

7.1. Threat: System Denial of Service Attacks

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

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

using the token mechanism.

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

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

7.2. Threat: Protocol Security

7.2.1. Threat: Authorization Mechanisms Compromised

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

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

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

7.2.2. Threat: Data Object Spoofing

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

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

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

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

8. IANA Considerations

This document does not have any IANA considerations.

9. Acknowledgments

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

Carsten Bormann

David Bryan

Dave Crocker

Yingjie Gu

David Harrington

Hongqiang (Harry) Liu

David McDysan

Borje Ohlman

Konstantinos Pentikousis

Martin Stiemerling

Richard Woundy

Ning Zong

10. References

10.1. Normative References

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

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

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

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

10.2. Informative References

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

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

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

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

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

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

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

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

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

A.1. Data Access Interface

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

A.2. Data Management Operations

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

A.3. Data Search Capability

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

A.4. Access Control Authorization

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

A.5. Resource Control Interface

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

A.6. Discovery Mechanism

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

A.7. Storage Mode

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

Appendix B. Hisotry

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

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

Authors' Addresses

Richard Alimi
Google

Email: ralimi@google.com

Akbar Rahman
InterDigital Communications, LLC

Email: akbar.rahman@interdigital.com

Dirk Kutscher
NEC

Email: dirk.kutscher@neclab.eu

Y. Richard Yang
Yale University

Email: yry@cs.yale.edu

Haibin Song
Huawei

Email: haibin.song@huawei.com

Kostas Pentikousis
Huawei

Email: k.pentikousis@huawei.com

DECADE
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

H. Song
N. Zong
Huawei
Y. Yang
Yale University
R. Alimi
Google

March 14, 2011

DECoupled Application Data Enroute (DECADE) Problem Statement
draft-ietf-decade-problem-statement-03

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 (the download traffic may increase because the in-network storage is likely much better connected). Traditional P2P and Web caches provide such storage, but they are complex (due to explicitly supporting individual P2P application protocols and cache refreshing mechanisms) and they do not allow users to manage access to content in the cache. For example, content providers cannot easily control cache access and resource usage policies to satisfy their own requirements, in the case when the content provider is also the user of in-network storage. This document discusses the introduction of in-network storage for P2P applications, shows the need for a standard protocol for accessing this storage, and identifies the scope of this protocol. The access protocol can also be used by other applications with similar requirements.

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

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

This Internet-Draft will expire on September 15, 2011.

Table of Contents

1. Introduction

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 the networks. Allowing P2P applications to store and retrieve data from inside networks can reduce traffic on the last-mile uplink, as well as backbone and transit links [I-D.weaver-alto-edge-caches].

P2P caches provide in-network storage and have been deployed in some networks. But 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 to effectively utilize in-network storage. In particular, P2P caches typically do not allow users to explicitly store content into in-network storage. They do not allow users to implement control over the content that has been placed into the in-network storage either.

Both of these challenges can be effectively addressed by using open, standard protocols to access in-network storage. P2P applications can store and retrieve content in the in-network storage, as well as control resources (e.g., network access, 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.

This document uses DECADE to refer to the protocol(s) developed or employed by the DECADE Working Group to provide these capabilities.

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.

With DECADE, P2P applications can still use their native protocols for signaling and data transport. However, they may use a standard protocol for data access incorporating in-network storage, and fall back to their native data transport protocols if in-network storage is not available or not used.

In essence, an open, standard protocol to access in-network storage provides an alternative mechanism for P2P application data access

that is decoupled from P2P application control and signaling. This decoupling leads to many advantages, which is explained further in Section 4.

And further, either the existing P2P cache or any new type of in-network storage should be deployed near the edge of the ISP's network so as to gain better performance.

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 network access to read/write data to applications. In-network storage may reduce upload/transit/backbone traffic and improve network application performance.

IAP (In-network storage Access Protocol): a standard protocol that is spoken between P2P applications and in-network storage. The protocol may also be used between entities implementing the in-network storage service. IAP may be a new protocol or existing protocol with extensions.

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.

Content Publisher: An entity that originates content.

3. The Problems

The emergence of peer-to-peer (P2P) as a major type of network application (esp. P2P file sharing and streaming apps) has led to substantial opportunities. The P2P paradigm can be utilized in designing highly scalable and robust applications at low cost, compared with traditional client-server paradigms. For example, CNN reported that P2P streaming by Octoshape played a major role in its distribution of the historical inauguration address of President Obama[Octoshape]. PPLive, one of the largest P2P streaming vendors, is able to distribute large-scale, live streaming programs to more than 2 million users with only a handful of servers[PPLive].

However, P2P applications also face substantial design challenges. A

particular problem facing P2P applications is the substantial stress that they place on the network infrastructure. Also, lack of infrastructure support can lead to unstable P2P application performance during peer churns and flash crowds. During a flash crowd, a large group of application users begin to access the same service during a very short period of time, which is a challenge to the system. Below we elaborate on the problems in more 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 (ISP). Multiple measurements (e.g., [ipoque.com]) have shown that P2P traffic has become a major type of traffic on some networks. Furthermore, network-agnostic peering (P2P transmission level) leads to unnecessary traversal across network domains or spanning the backbone of a network, leading to network inefficiency [RFC5693].

Recently, the IETF Application Layer Traffic Optimization (ALTO) Working Group was formed to provide 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 pure P2P system. On the other hand, it is reported that P2P traffic is becoming the dominating traffic on the access networks of some networks, reaching as high as 50-60% at the down-links and 60-90% at the uplinks ([DCIA], [ICNP], [ipoque.P2P_survey.], [P2P_file_sharing]). 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 potential (relatively) slow upstream transmission. Similarly, the performance of real-time P2P applications may be

affected by potential (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. For example, in [I-D.weaver-alto-edge-caches], the author demonstrates clearly the potential benefits of introducing in-network storage to improve network efficiency and thus reduce network infrastructure stress.

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 be evolving 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, high cost, and low reliability.

Third, many P2P applications use proprietary protocols or support end-to-end encryption. This can render P2P caches ineffective.

3.3. Ineffective integration of P2P applications

As P2P applications evolve, it is becoming increasingly clear that they will need in-network resources to provide positive user experiences. 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.

A requirement by some P2P applications in using in-network infrastructural resources, however, is 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. They continue to fine-tune such algorithms. In other words, in-network storage should export basic mechanisms and allow as much flexibility as possible to P2P applications to implement specific policies. 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. DECADE as an In-network Storage Capability

The objective of this working group is to design DECADE, which primarily consists of an in-network storage access protocol (IAP) to address the problems discussed in the preceding section.

DECADE will provide access to storage and data transport services in the network to P2P applications to improve their efficiency and reduce the stress on the network infrastructure. Unlike the existing P2P caching architecture, DECADE is a standard interface for various P2P applications (both content publishers and end users) to access in-network storage. This decoupling of P2P data access from P2P application control and signaling reduces the complexity of in-network storage services. Furthermore, DECADE provides basic access mechanisms and allows P2P applications to implement flexible policies to create an ecosystem for application innovation and various business goals. Besides that, it also improves the availability of P2P contents because the in-network storage is always-on.

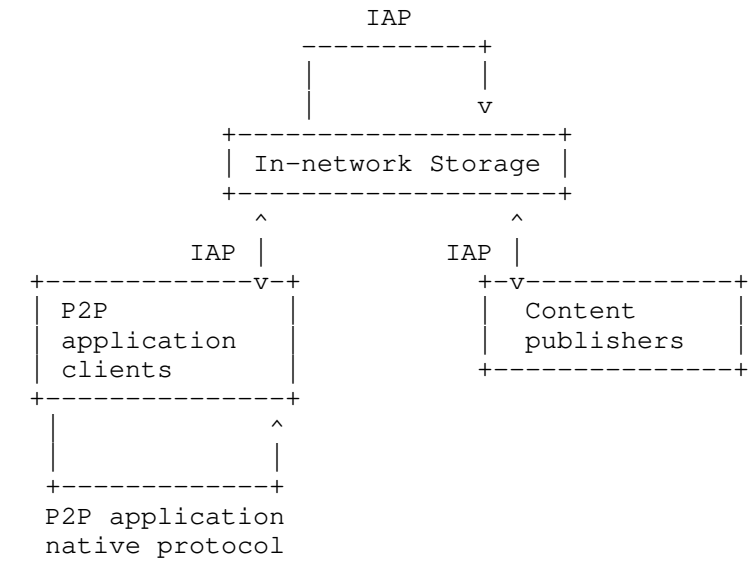


Figure 1 Overview of protocol interaction between DECADE elements

Specifically, the main component of DECADE is the in-network storage access protocol (IAP), which is a standard, P2P-application-agnostic protocol for different P2P applications to access in-network storage. IAP defines a set of commands that P2P application elements can issue to in-network storage to store and retrieve data. IAP includes the following functionality:

(1) Data access provides read/write by users (e.g., P2P application clients and content publishers) to the corresponding in-network storage and between entities implementing the in-network storage;

(2) Authorization implements access control to resources provided by in-network storage;

(3) Resource control allows users to implement application policies for the corresponding in-network storage.

While DECADE will focus on P2P applications, the solution is expected to be applicable in other contexts with similar requirements.

4.1. Data access

P2P application clients use the IAP protocol to read data from an in-network storage, store data to an in-network storage, or remove data from an in-network storage. The data could be of various types. Existing protocols will be used wherever possible and appropriate to

support DECADE's requirements. In particular, data storage, retrieval, and management may be provided by existing IETF protocols. The WG will not limit itself to a single data transport protocol since different protocols may have varying implementation costs and performance tradeoffs. However, to keep interoperability manageable, a small number of specific, targeted, data transport protocols will be identified and used. If a protocol is found to be suitable but does not fully meet the requirements, then the protocol may need to be extended. The following considerations should be taken into account, although there might be trade-offs among these considerations.

- o The protocol(s) should support deployments with a very large number of users without substantial increase to operational complexity for the storage provider.

- o The protocol(s) should be easy for application integration, whether they want to use it for P2P applications (e.g. file-sharing or streaming) or for other content distribution applications.

4.2. Authorization

DECADE ensures that access to a user's in-network storage is subject to authorization by that user. The authorization can take into account the user trying to access, the content, the time period, etc.

4.3. Resource control

A user uses the IAP protocol to manage the resources on in-network storage that can be used by other peers, e.g., network access to the storage or network connections themselves. The resource control policies could be based on individual remote peers or a whole application.

5. Usage Scenarios

Usage scenarios are described from two perspectives. First, we introduce high-level use cases showing how P2P applications may utilize in-network storage. Second, we show how in more detail how users exchange data using IAP.

5.1. BitTorrent

BitTorrent may be integrated with DECADE to be more network efficient and reduce the bandwidth consumed on ISP networks. When a BitTorrent client uploads a block to peers, the block traverses the last-mile

uplink once for each peer. With DECADE, 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.

We now describe in more detail how BitTorrent can utilize DECADE. For illustration, we assume that both the BitTorrent client (A) and its peer (B) utilize in-network storage. When A requests a block, it indicates that it would like the block stored in its in-network storage and provides the necessary access control. Instead of sending the 'piece' message with the desired block, peer B replies with a 'redirect' message indicating that the content should be retrieved from its own in-network storage and provides the necessary access control. If the peer B had not previously stored the content in its in-network storage, it uploads the block. A instructs its in-network storage to download the block from B's in-network storage, and finally A itself retrieves the block.

Note that this requires extensions to the BitTorrent protocol. While there are multiple ways to do so, this example assumes the native BitTorrent 'request' message is extended to carry additional information and that a new 'redirect' message is added. Upload and download to/from in-network storage uses the standard IAP protocol.

This example has illustrated how utilizing DECADE can increase BitTorrent's network efficiency. First, notice that peer B does not utilize any uplink bandwidth if the block was already present in its in-network storage. Second, notice that the block is downloaded directly into A's in-network storage. When A wishes to share the block with another peer (say, peer C) that supports DECADE, it may upload directly from its in-network storage, again avoiding usage of the last-mile uplink.

Redirection to a DECADE server does not only need to come from a peer. In this case, in order to avoid the connectivity issue brought by NATs, B can also attach its in-network storage address in the message to its tracker. When A sends the content request message with the content ID to the tracker, the tracker replies with B's in-network storage address and the BitMap info. Then A sends a request using IAP protocol to B's in-network storage for the pieces of this content, with the unique identity of the content in the storage.

This technique can be applied to other P2P applications as well. Since P2P applications use a standard for communicating with in-network storage, they no longer require in-network storage to explicitly support their protocol. P2P applications retain the ability to explicitly manage which content is placed in in-network storage, as well as access and resource control polices.

5.2. 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.

Consider a case where the content publisher owns an in-network storage account within ISP A. If there are multiple P2P peers within ISP A, the content publisher may utilize DECADE to distribute content to the peers.

First, the content publisher stores a block in the in-network storage, and then sends to each peer in ISP A the block's identifier and necessary access control. Second, each peer may then download from the content publisher's in-network storage.

In this example, the block is distributed in a more network efficient way (the content only traverses the ISP's interdomain link once), while the content publisher retains explicit control over access to the content placed in its own storage. The content publisher can remove content from its in-network storage when it is stale or needs to be replaced, and grant access and resources to only the desired peers.

Note that content publishers and individual peers can each use in-network storage. For example, after downloading content from the content publisher's in-network storage, peers may each utilize their own in-networks storage similar to the usage scenario in Section 5.1. This can have the benefit of increased network efficiency, while content publishers and peers still retain control over content placed in their own in-network storage.

If it desires, a content publisher may still apply DRM to the payload. This is independent of any authentication or authorization provided by DECADE.

5.3. CDN/P2P hybrid

Some applications use a 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.

DECADE may be beneficial to hybrid CDN/P2P applications as well. However, if only the endhost can store content in the DECADE server,

the content must be downloaded and then uploaded over the last-mile access link before another peer may retrieve it from a DECADE server. Thus, in this deployment scenario, it may be advantageous for a content publisher or CDN provider to store content on DECADE servers.

5.4. Data Transfer Scenarios

The previous usage scenarios have utilized the ability for peers to transfer data by storing and retrieving from in-network storage. This section describes in further detail an example solution of how DECADE can provide this capability. It is important to note that this example is provided to illustrate more concretely the capabilities provided by DECADE, and is not intended to reflect any particular solution methodology or protocol(s) developed by the DECADE Working Group.

In this section, we consider the case of a user B (the receiver) requesting data from user A (the sender). We use Sa to denote User A's storage account, and Sb to denote User B's storage account. Each user independently decides if its in-network storage account is used, so there are four cases.

When a user indicates that it wishes to use its in-network storage, it provides an access control token the other user. The token is sent using the application's protocol. To simplify the illustration, we omit details of the access control from the detailed scenarios below.

5.4.1. Both Sender and Receiver Accounts are Used

This scenario is illustrated in Figure 2. B first requests an object from A using the application protocol indicating it wishes the object to be stored in Sb. A responds using the application protocol indicating that B should download the object from Sa. B sends a IAP request to Sb indicating that the object should be downloaded from Sa. Sb uses IAP to download from Sa, and finally, B downloads the object from Sb (also using IAP).

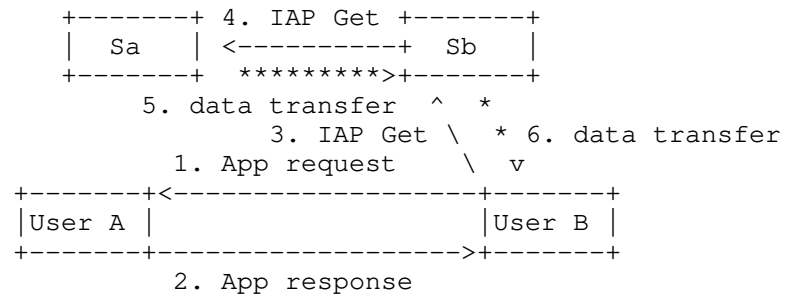


Figure 2: Usage Scenario 1 (Sender and receiver Accounts used)

5.4.2. Only Sender's Storage Account is Used

This scenario is illustrated in Figure 3. B requests an object from A using the application protocol. A responds using the application protocol indicating that B should download the object from Sa. Finally, B sends a IAP request to Sa to download the object.

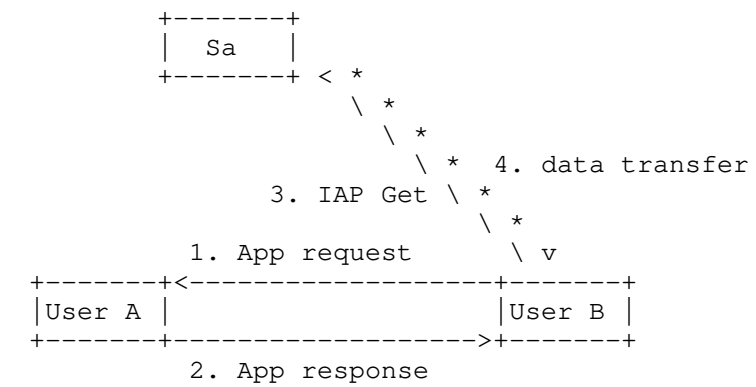


Figure 3: Usage Scenario 2 (Sender account used)

5.4.3. Only Receiver's Storage Account is Used

This scenario is illustrated in Figure 4. B requests an object from A using the application protocol indicating that it wishes the object to be stored in Sb. A stores the object in Sb (using IAP), and responds to B (using the application protocol) that it should download from Sb. B uses IAP to download the object from Sb.

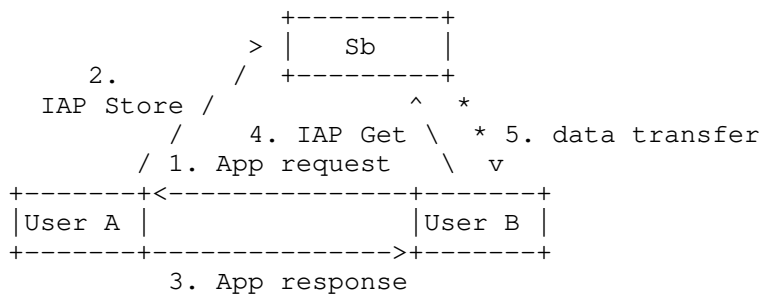


Figure 4: Usage Scenario 3 (Receiver account used)

5.4.4. No Storage Accounts are Used

This scenario is illustrated in Figure 5. In this scenario, the application protocol is used directly to send data. This scenario applies with one of the peers does not support IAP, or neither of the peers are using in-network storage.

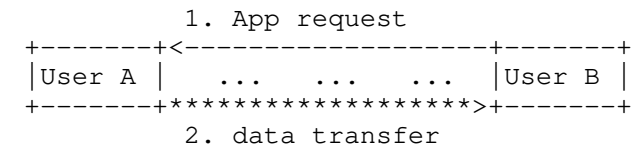


Figure 5: Usage Scenario 4 (No storage accounts used)

6. Security Considerations

There are multiple security considerations. We focus on two in this section.

6.1. Denial of Service Attacks

Without access control or resource control, an attacker can try to consume a large portion of the in-network storage, or exhaust the connections of the in-network storage to commit a Denial of Service (DoS) attack. Thus, access control and resource control mechanisms are mandatory. A resource control mechanism should be used to allow a user to allocate the resource in its in-network storage account to be utilized by other clients.

6.2. Copyright and Legal Issues

Copyright and other laws may prevent the distribution of certain content in various localities. While in-network storage operators

may adopt system-wide ingress or egress filters to implement necessary policies for storing or retrieving content, and applications may still apply DRM to the data stored in the network storage, the specification and implementation of such policies (e.g., filtering and DRM) is outside of the scope of this working group.

7. IANA Considerations

There is no IANA consideration with this document.

8. Acknowledgments

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

David Bryan

Kar Ann Chew

Roni Even

Lars Eggert

Yingjie Gu

Francois Le Faucheur

Hongqiang Liu

Tao Ma

Borje Ohlman

Akbar Rahman

Yu-shun Wang

Richard Woundy

Yunfei Zhang

9. Informative References

[ipoque.com]
"http://www.ipoque.com/resources/internet-studies/

internet-study-2008_2009".

[RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, October 2009.

[I-D.weaver-alto-edge-caches] Weaver, N., "Peer to Peer Localization Services and Edge Caches", draft-weaver-alto-edge-caches-00 (work in progress), March 2009.

[I-D.ietf-ppsp-problem-statement] Zhang, Y., Zong, N., Camarillo, G., Seng, J., and Y. Yang, "Problem Statement of P2P Streaming Protocol (PPSP)", draft-ietf-ppsp-problem-statement-01 (work in progress), January 2011.

[DCIA] http://www.dcia.info, "Distributed Computing Industry Association".

[ipoque.P2P_survey.] "Emerging Technologies Conference at MIT", Sept. 2007.

[P2P_file_sharing] Parker, A., "The true picture of peer-to-peer filesharing", July 2004.

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

[PPLive] "http://www.synacast.com/products/".

[ICNP] Wu, H., "Challenges and opportunities of Internet developments in China, ICNP 2007 Keynote", Oct. 2007.

Appendix A. Other Related Work in IETF

Note that DECADE is independent of current IETF work on P2P, e.g. P2PSIP, ALTO, and PPSP. For example, peers discovered by either RELOAD or ALTO can all use DECADE to share data.

The Peer to Peer Streaming Protocol effort in the IETF is investigating specification of signaling protocols (called PPSP protocols) for multiple types of 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

document [I-D.ietf-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. A DECADE service can be used for any application transparently to the DECADE 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. So 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 considered 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 considered 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. say because an arrangement has been put in place between the content provider and the considered 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 (using IAP protocol) by PPSP peers (both off-net end-users and in-network CDN nodes).

PPSP and DECADE jointly to 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 to require more information in PPSP tracker protocol, e.g., the mobile node can indicate its DECADE in-network proxy to PPSP tracker and the following requesting peer can finish its data transfer with the DECADE proxy with IAP.

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

DECADE
Internet-Draft
Intended status: Informational
Expires: November 8, 2012

H. Song
N. Zong
Huawei
Y. Yang
Yale University
R. Alimi
Google

May 7, 2012

DECoupled Application Data Enroute (DECADE) Problem Statement
draft-ietf-decade-problem-statement-06

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.

Table of Contents

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 chose 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 polices. 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

Lars Eggert

Francois Le Faucheur

Adrian Farrel

Yingjie Gu

David Harrington

Leif Johansson

Hongqiang Liu

Tao Ma

Borje Ohlman

Akbar Rahman

Robert Sparks

Peter Saint-Andre

Sean Turner

Yu-shun Wang

Richard Woundy

Yunfei Zhang

8. Informative References

[Internet_Study_2008-2009]
"Internet Study 2008/2009", <http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009>.

[RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, October 2009.

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

[Data_Lockers]
Yang, Y., "Open Content Distribution using Data Lockers",

<http://cs-www.cs.yale.edu/homes/yry/projects/p4p/open-data-lockers-nov-2010-coxnet.pdf>.

[I-D.ietf-p2psip-base]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-21 (work in progress), March 2012.

[DCIA] http://www.dcia.info, "Distributed Computing Industry Association".

[ipoque.P2P_survey.]
"Emerging Technologies Conference at MIT", Sept. 2007.

[P2P_file_sharing]
Parker, A., "The true picture of peer-to-peer filesharing", July 2004.

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

[PPLive] "http://www.synacast.com/products/".

[ICNP] Wu, H., "Challenges and opportunities of Internet developments in China, ICNP 2007 Keynote", Oct. 2007.

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

DECADE
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

Y. Gu
Huawei
D. Bryan
Cogent Force, LLC / Huawei
Y. Yang
Yale University
R. Alimi
Google

March 14, 2011

DECADE Requirements
draft-ietf-decade-reqs-01

Abstract

The target of DECoupled Application Data Enroute (DECADE) is to provide an open and standard in-network storage system for applications, primarily P2P applications, to store, retrieve and manage their data. This draft enumerates and explains requirements, not only for store and retrieve, but also for data management, access control and resource control, that should be considered during the design and implementation of a DECADE system. These are requirements on the entire system; some of the requirements may eventually be implemented by an existing protocol with/without some extensions (e.g., the data transport level). A user of DECADE as a complete architecture would be guaranteed complete functionality.

Requirements Language

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

Status of this Memo

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

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

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

This Internet-Draft will expire on September 15, 2011.

Copyright Notice

Table of Contents

1. Introduction

The object of DECoupled Application Data Enroute (DECADE) is to provide an open and standard in-network storage system for applications, primarily applications that could be implemented using a content distribution paradigm, where data is broken in to one or more chunks and then distributed. This may already include many types of applications including P2P applications, IPTV, and VoD. Instead of always transferring data directly from a source/owner client to a requesting client, the source/owner client can store and manage its content on its in-network storage. The requesting client can get the address of the in-network storage pertaining to the source/owner client and retrieve data from the storage.

This draft enumerates and explains the rationale behind SPECIFIC requirements on the protocol design and on any data store implementation that may be used to implement DECADE servers that should be considered during the design and implementation of a DECADE system. As such, it DOES NOT include general guiding principals. General design considerations, explanation of the problem being addressed, and enumeration of the types of applications to which DECADE may be suited is not considered in this document. For general information, please see the problem statement [I-D.ietf-decade-problem-statement] and architecture drafts.

This document enumerates the requirements to enable target applications to utilize in-network storage. In this context, using storage resources includes not only basic capabilities such as storing and retrieving data, and managing data, but also (1) controlling access for particular remote clients with which it is sharing data and (2) controlling the resources used by remote clients when they access data.

2. Terminology and Concepts

This document uses terms defined in [I-D.ietf-decade-problem-statement]. In particular, IAP refers to the In-network storage Access Protocol, which is the protocol used to communicate between a DECADE client and DECADE server (in-network storage) for access control and resource control.

This document also defines additional terminology:

Target Application: An application (typically installed at end-hosts) with the ability to explicitly control usage of network and/or storage resources to deliver contents to a large number of users. This includes scenarios where multiple applications or entities

cooperate, such as with P2P/CDN hybrid architectures. Such applications distribute large contents (e.g., a large file, or video stream) by dividing the contents into smaller blocks for more flexible distribution (e.g., multipath). The distributed content is typically immutable (though it may be deleted). We use the term Target Application to refer to the type of applications that are explicitly (but not exclusively) supported by DECADE.

3. Requirements Structure

The DECADE protocol is intended to sit between Target Applications and a back-end storage system. In the development of DECADE, it must be made clear that the intention is to NOT develop yet another storage system, but rather to create a protocol that enables Target Applications to make use of storage within the network, leaving specific storage system considerations to the implementation of the DECADE servers as much as possible. For this reason, we have divided the requirements into two categories:

- Protocol Requirements: Protocol requirements for Target Applications to make use of in-network storage within their own data dissemination schemes. Development of these requirements is guided by a study of data access, search and management capabilities used by Target Applications. These requirements may be met by a new protocol to be defined within the DECADE Working Group.

- Storage Requirements: Functional requirements necessary for the back-end storage system employed by the DECADE server. Development of these requirements is guided by a study of the data access patterns used by Target Applications. These requirements should be met by the underling data transport used by DECADE.

It should also be made clear that the approach is to make DECADE a simple protocol, while still enabling its usage within many Target Applications. For this reason, and to further reinforce the distinction between DECADE and a storage system, in some cases we also highlight the non-requirements of the protocol. These non-requirements are intended to capture behaviors that will NOT be assumed to be needed by DECADE's Target Applications and hence not present in the DECADE protocol.

Finally, some implementation considerations are provided, which while strictly are not requirements, are intended to provide guidance and highlight potential points of concern that need to be considered by the protocol developers, and later by implementors.

4. Protocol Requirements

4.1. Requirements

4.1.1. Overall Protocol Requirements

4.1.1.1. Cross-platform Access

REQUIREMENT(S): If DECADE supports the ability to store metadata associated with data objects, the DECADE protocol(s) MUST transmit any metadata using an operating system-independent and architecture-independent format.

RATIONALE: If DECADE supports the possibility for storing metadata (e.g., a description, uploaded date, object size, or access control list), a possible use for the metadata is to help a DECADE client locate a desired data object. Data objects may be stored by DECADE clients running on various platforms. To enable metadata to be readable regardless of its source it must be transmitted to and from the DECADE server in a standard format.

4.1.1.2. Connectivity Concerns

4.1.1.2.1. NATs and Firewalls

REQUIREMENT(S): DECADE SHOULD be usable across firewalls and NATs without requiring additional network support (e.g., Application-level Gateways).

RATIONALE: Firewalls and NATs are widely used in the Internet today, both in ISP networks and within households. Deployment of DECADE must not require modifications to such devices (beyond, perhaps, reconfiguration). Note that this requirement applies to both any new protocol developed by the DECADE Working Group and any data transport used with DECADE.

4.1.1.2.2. Connections to Clients

REQUIREMENT(S): DECADE SHOULD require that network connections be made from DECADE clients to DECADE servers (i.e., not to the DECADE client).

RATIONALE: Many household networks and operating systems have firewalls and NATs configured by default. To ease deployment by avoiding configuration changes and help mitigate security risks, DECADE should not require clients to listen for any incoming network connections (beyond what is required by any other already-deployed application).

4.1.1.3. Error and Failure Conditions

4.1.1.3.1. Overload Condition

REQUIREMENT(S): In-network storage, which is operating close to its capacity limit (e.g., too busy servicing other requests), MUST be able to reject requests and not be required to generate responses to additional requests.

RATIONALE: When in-network storage is operating at a limit where it may not be able to process additional requests, it should not be required to generate responses to such additional requests. Forcing the in-network storage to do so can impair its ability to service existing requests.

4.1.1.3.2. Insufficient Resources

REQUIREMENT(S): DECADE MUST support an error condition indicating to a DECADE client that resources (e.g., storage space) were not available to service a request (e.g., storage quota exceeded when attempting to store data).

RATIONALE: The currently-used resource levels within the in-network storage are not locally-discoverable, since the resources (disk, network interfaces, etc) are not directly attached. In order to allocate resources appropriately amongst remote clients, a client must be able to determine when resource limits have been reached. The client can then respond by explicitly freeing necessary resources or waiting for such resources to be freed.

4.1.1.3.3. Unavailable and Deleted Data

REQUIREMENT(S): DECADE MUST support error conditions indicating that (1) data was rejected from being stored, (2) deleted, or (3) marked unavailable by a storage provider.

RATIONALE: Storage providers may require the ability to (1) avoid storing, (2) delete, or (3) quarantine certain data that has been identified as illegal (or otherwise prohibited). DECADE does not indicate how such data is identified, but applications using DECADE should not break if a storage provider is obligated to enforce such policies. Appropriate error conditions should be indicated to applications.

4.1.1.3.4. Redirection

REQUIREMENT(S): DECADE SHOULD support the ability for a DECADE server to redirect requests to another DECADE server. This may be in response to either an error or failure condition, or to support capabilities such as load balancing.

RATIONALE: A DECADE server may opt to redirect requests to another server to support capabilities such as load balancing, or if the implementation decides that another DECADE server is in a better position to handle the request due to either its location in the network, server status, or other consideration.

4.1.2. Transfer and Latency Requirements

4.1.2.1. Low-Latency Access

REQUIREMENT(S): DECADE SHOULD provide "low-latency" access for application clients. DECADE MUST allow clients to specify at least two classes of services for service: lowest possible latency and latency non-critical.

RATIONALE: Some applications may have requirements on delivery time (e.g., live streaming [PPLive]). The user experience may be unsatisfactory if the use of in-network storage results in lower performance than connecting directly to remote clients over a low-speed, possibly congested uplink. Additionally, the overhead required for control-plane operations in DECADE must not cause the latency to be higher than for a low-speed, possibly congested uplink. While it is impossible to make a guarantee that a system using in-network storage will always outperform a system that does not for every transfer, the overall performance of the system should be improved compared with direct connections, even considering control overhead.

4.1.2.2. Indirect Transfer

REQUIREMENT(S): DECADE MUST allow a user's in-network storage to directly fetch from other user's in-network storage.

RATIONALE: As an example, a requesting remote client may get the address of the storage pertaining to the source/owner client and then tell its own in-network storage to fetch the content from the source-owner's in-network storage. This helps to avoid extra transfers across ISP network links where possible.

4.1.2.3. Data Object Size

REQUIREMENT(S): DECADE MUST allow for efficient data transfer of small objects (e.g., 16KB) between a DECADE client and in-network storage with minimal additional latency required by the protocol.

RATIONALE: Though Target Applications are frequently used to share large amounts of data (e.g., continuous streams or large files), the data itself is typically subdivided into smaller chunks that are transferred between clients. Additionally, the small chunks may have requirements on delivery time (e.g., in a live-streaming application). DECADE must enable data to be efficiently transferred amongst clients at this granularity.

4.1.2.4. Communication among In-network Storage Elements

REQUIREMENT(S): DECADE SHOULD support the ability for two in-network storage elements in different administrative domains to store and/or retrieve data directly between each other. If such a capability is supported, this MAY be the same (or a subset or extension of) as the IAP used by clients to access data.

RATIONALE: Allowing server-to-server communication can reduce latency in some common scenarios. Consider a scenario when a DECADE client is downloading data into its own storage from another client's in-network storage. One possibility is for the client to first download the data itself, and then upload it to its own storage. However, this causes unnecessary latency and network traffic. Allowing the data to be downloaded from the remote in-network storage into the client's own in-network storage can alleviate both.

4.1.3. Data Access Requirements

4.1.3.1. Reading/Writing Own Storage

REQUIREMENT(S): DECADE MUST support the ability for a DECADE client to read data from and write data to its own in-network storage.

RATIONALE: Two basic capabilities for any storage system are reading and writing data. A DECADE client can read data from and write data to in-network storage space that it owns.

4.1.3.2. Access by Other Users

REQUIREMENT(S): DECADE MUST support the ability for a user to apply access control policies to users other than itself for its storage. The users with whom access is being shared can be under a different administrative domain than the user who owns the in-network storage. The authorized users may read from or write to the user's storage.

RATIONALE: Endpoints in Target Applications may be located across multiple ISPs under multiple administrative domains. Thus, to be useful by Target Applications, DECADE allows a user to specify access control policies for users that may or may not be known to the user's storage provider.

4.1.3.3. Negotiable Data Protocol

REQUIREMENT(S): DECADE MUST support the ability for a DECADE client to negotiate with its in-network storage about which protocol it can use to read data from and write data to its In-network storage. DECADE MUST specify at least one mandatory protocol to be supported by implementations; usage of a different protocol may be selected via negotiation.

RATIONALE: Since typical data transport protocols (e.g., NFS and WebDAV) already provide read and write operations for network storage, it may not be necessary for DECADE to define such operations in a new protocol. However, because of the particular application requirements and deployment considerations, different applications may support different protocols. Thus, a DECADE client must be able to select an appropriate protocol also supported by the in-network storage. This requirement also follows as a result of the requirement of Separation of Control and Data Operations (Section 4.1.3.4).

4.1.3.4. Separation of Data Operations from Application Control

REQUIREMENT(S): The DECADE IAP MUST only provide a minimal set of core operations to support diverse policies implemented and desired by Target Applications.

RATIONALE: Target Applications support many complex behaviors and diverse policies to control and distribute data, such as (e.g., search, index, setting permissions/passing authorization tokens). Thus, to support such Target Applications, these behaviors must be logically separated from the data transfer operations (e.g., retrieve, store). Some minimal overlap (for example obtaining credentials needed to encrypt or authorize data transfer using control operations) may be required to be directly specified by DECADE.

4.1.4. Data Management Requirements

4.1.4.1. Agnostic of reliability

REQUIREMENT(S): DECADE SHOULD remain agnostic of reliability/ fault-tolerance level offered by storage provider.

RATIONALE: Providers of a DECADE service may wish to offer varying levels of service for different applications/users. However, a single compliant DECADE client should be able to use multiple DECADE services with differing levels of service.

4.1.4.2. Time-to-live for Stored Data

REQUIREMENT(S): DECADE MUST support the ability for a DECADE client to indicate a time-to-live value (or expiration time) indicating a length of time until particular data can be deleted by the in-network storage element.

RATIONALE: Some data stored by a DECADE client may be usable only within a certain window of time, such as in live-streaming P2P applications. Providing a time-to-live value for stored data (e.g., at the time it is stored) can reduce management overhead by avoiding many 'delete' commands sent to in-network storage. The in-network storage may still keep the data in cache for bandwidth optimization. But this is guided by the privacy policy of the DECADE provider.

4.1.4.3. Offline Usage

REQUIREMENT(S): DECADE MAY support the ability for a user to provide authorized access to its in-network storage even if the user has no DECADE applications actively running or connected to the network.

RATIONALE: If an application desires, it can authorize remote clients to access its storage even after the application exits or network connectivity is lost. An example use case is mobile scenarios, where a client can lose and regain network connectivity very often.

4.1.5. Resource Control

4.1.5.1. Multiple Applications

REQUIREMENT(S): DECADE SHOULD support the ability for users to define resource sharing policies for multiple applications (DECADE clients) being run/managed by the user.

RATIONALE: A user may own in-network storage and share the in-network storage resources amongst multiple applications. For example, the user may run a video-on-demand application and a live-streaming (or even two different live-streaming applications) application which both make use of the user's in-network storage. The applications may be running on different machines and may not directly communicate. Thus, DECADE should enable the user to determine resource sharing policies between the applications.

One possibility is for a user to indicate the particular resource sharing policies between applications out-of-band (not using a standard protocol), but this requirement may manifest itself in passing values over IAP to identify individual applications. Such identifiers can be either user-generated or server-generated and do not need to be registered by IANA.

4.1.5.2. Per-Remote-Client, Per-Data Control

REQUIREMENT(S): A DECADE client MUST be able to assign resource quotas to individual remote clients for reading from and writing particular data to its in-network storage within a particular range of time. The DECADE server MUST enforce these constraints.

RATIONALE: Target Applications can rely on control of resources on a per-remote-client or per-data basis. For example, application policy may indicate that certain remote clients have a higher bandwidth share for receiving data [LLSB08]. Additionally, certain data (e.g., chunks) may be distributed with a higher priority. As another example, when allowing a remote client to write data to a user's in-network storage, the remote client may be restricted to write only a certain amount of data. Since the client may need to manage multiple clients accessing its data, it should be able to indicate the time over which the granted resources are usable. For example, an expiration time for the access could be indicated to the server after which no resources are granted (e.g., indicate error as access denied).

4.1.5.3. Server Involvement

REQUIREMENT(S): A DECADE client MUST be able to indicate to a DECADE server, without itself contacting the server, resource control policies for remote clients' requests.

RATIONALE: One important consideration for in-network storage elements is scalability, since a single storage element may be used to support many users. Many Target Applications use small chunk sizes and frequent data exchanges. If such an application employed resource control and contacted the in-network storage element for each data exchange, this could present a scalability challenge for the server as well as additional latency for clients.

An alternative is to let requesting users get the resource control policies and users can then present the policy to the storage directly. This can result in reduced messaging handled by the in-network storage.

4.1.6. Authorization

4.1.6.1. Per-Remote-Client, Per-Data Read Access

REQUIREMENT(S): A DECADE Client MUST be able to control which individual remote clients are authorized to read particular data stored on its in-network storage.

RATIONALE: A Target Application can control certain application-level policies by sending particular data (e.g., chunks) to certain remote clients. It is important that remote clients not be able to circumvent such decisions by arbitrarily reading any currently-stored data in in-network storage.

4.1.6.2. Per-User Write Access

REQUIREMENT(S): A DECADE Client MUST be able to control which individual remote clients are authorized to store data into its in-network storage.

RATIONALE: The space managed by a user in in-network storage can be a limited resource. At the same time, it can be useful to allow remote clients to write data directly to a user's in-network storage. Thus, a DECADE client should be able to grant only certain remote clients this privilege.

Note that it is not (currently) a requirement to check that a remote client stores a particular set of data (e.g., the check that a remote client writes the expected chunk of a file). Enforcing this as a requirement would require a client to know

which data is expected (e.g., the full chunk itself or a hash of the chunk) which may not be available in all applications. Checking for a particular hash could be considered as a requirement in the future that could optionally be employed by applications.

4.1.6.3. Authorization Checks

REQUIREMENT(S): In-network storage MUST check the authorization of a client before it executes a supplied request. The in-network storage MAY use optimizations to avoid such authorization checks as long as the enforced permissions are the the same.

RATIONALE: Authorization granted by a DECADE client are meaningless unless unauthorized requests are denied access. Thus, the in-network storage element must verify the authorization of a particular request before it is executed.

4.1.6.4. Credentials Not IP-Based

REQUIREMENT(S): Access MUST be able to be granted on other credentials than the IP address

RATIONALE: DECADE clients may be operating on hosts without constant network connectivity or without a permanent attachment address (e.g., mobile devices). To support access control with such hosts, DECADE servers must support access control policies that use information other than IP addresses.

4.1.6.5. Server Involvement

REQUIREMENT(S): A DECADE client MUST be able to indicate, without contacting the server itself, access control policies for remote clients' requests.

RATIONALE: See discussion in Section 4.1.5.3.

5. Storage Requirements

5.1. Requirements

5.1.1. Explicit Deletion of Stored Data

REQUIREMENT(S): DECADE MUST support the ability for a DECADE client to explicitly delete data from its own in-network storage. DECADE MAY have an overwrite flag indicating that an object with the same name should be replaced.

RATIONALE: A DECADE client may continually be writing data to its in-network storage. Since there may be a limit (e.g., imposed by the storage provider) to how much total storage can be used, some data may need to be removed to make room for additional data. A DECADE client should be able to explicitly remove particular data. This may be implemented using existing protocols.

5.1.2. Multiple writing

REQUIREMENT(S): DECADE MUST NOT allow multiple writers for the same object. Implementations raise an error to one of the writers.

RATIONALE: This avoids data corruption in a simple way while remaining efficient.

5.1.3. Multiple reading

REQUIREMENT(S): DECADE MUST allow for multiple readers for an object.

RATIONALE: One characteristic of Target Applications is the ability to upload an object to multiple clients. Thus, it is natural for DECADE to allow multiple readers to read the content concurrently.

5.1.4. Reading before completely written

REQUIREMENT(S): DECADE MAY allow readers to read from objects before they have been completely written.

RATIONALE: Some Target Applications (in particular, P2P streaming) can be sensitive to latency. A technique to reduce latency is to remove store-and-forward delays for data objects (e.g., make the object available before it is completely stored). Appropriate handling for error conditions (e.g., a disappearing writer) needs to be specified.

5.1.5. Hints concerning usage stored data

REQUIREMENT(S): DECADE MAY allow writers of new objects to indicate specific hints concerning how the objects are expected to be used (e.g., access frequency or time to live).

RATIONALE: Different Target Applications may have different usage patterns for objects stored at in-network storage. For example, a P2P live streaming application may indicate to a DECADE server that the objects are expected to have a shore time-to-live, but read frequently. The DECADE server may then opt to store the objects in memory instead of in disk.

5.1.6. Writing model

REQUIREMENT(S): DECADE MUST provide at least a writing model (while storing an object) that appends data to data already stored.

RATIONALE: Depending on the object size (e.g., chunk size) used by a Target Application, the application may need to send data to the DECADE server in multiple packets. To keep implementation simple, the DECADE must at least support the ability to write the data sequentially in the order received. Implementations MAY allow application to write data in an object out-of-order (but MUST NOT overwrite ranges of the object that have already been stored).

5.1.7. Storage Status

REQUIREMENT(S): A DECADE client MUST be able to retrieve current resource usage (including list of stored data), resource quotas, and access permissions for its in-network storage. The returned information MUST include resource usage resulting from the client's own usage and usage by other clients that have been authorized to read/write objects or open connections to that client's storage.

RATIONALE: The resources used by a client are not directly-attached (e.g., disk, network interface, etc). Thus, the client cannot locally determine how such resources are being used. Before storing and retrieving data, a client should be able to determine which data is available (e.g., after an application restart). Additionally, a client should be able to determine resource availability to better allocate them to remote clients.

5.2. Non-Requirements

5.2.1. No ability to update

REQUIREMENT(S): DECADE SHOULD NOT provide ability to update existing objects. That is, objects are immutable once they are stored.

RATIONALE: Reasonable consistency models for updating existing objects would significantly complicate implementation (especially if implementation chooses to replicate data across multiple servers). If a user needs to update a resource, it can store a new resource and then distribute the new resource instead of the old one.

6. Implementation Considerations

The intent of this section is to collect discussion items and implementation considerations that have been discovered as this requirements document has been produced. The content of this section will be migrated to an appropriate place as the document and the Working Group progress.

6.1. Resource Scheduling

The particular resource scheduling policy may have important ramifications on the performance of applications. This document has explicitly mentioned simultaneous support for both low-latency applications and latency-tolerant applications.

Denial of Service attacks may be another risk. For example, rejecting new requests due to overload conditions may introduce the ability to perform a denial of service attack depending on a particular DECADE server's scheduling implementation and resource allocation policies.

6.2. Removal of Duplicate Records

There are actually two possible scenarios here. The first is the case of removing duplicates within one particular DECADE server (or logical server). While not a requirement, as it does not impact the protocol and is technically not noticeable on message across the wire, a DECADE server may implement internal mechanisms to monitor for duplicate records and use internal mechanisms to prevent duplication of internal storage.

The second scenario is removing duplicates across a distributed set of DECADE servers. This is a more difficult problem, and if the group decides to support this capability, it may require protocol support. See Section 7.2 for more details.

7. Discussion and Open Issues

7.1. Discussion

Sometimes, several logical in-network storages could be deployed on the same physical network device. In this case, in-network storages on the same physical network device can communicate and transfer data through internal communication messages. However in-network storages deployed on different physical network devices SHOULD communicate with in-network storage Access Protocol (IAP).

To provide fairness among users, the in-network storage provider should assign resource (e.g., storage, bandwidth, connections) quota for users. This can prevent a small number of clients from occupying large amounts of resources on the in-network storage, while others starve.

7.2. Open Issues

Gaming of the Resource Control Mechanism: There has been some discussion of how applications may be able game the scheduling system by manipulating the resource control mechanism, for example by specifying many small peers to increase total throughput. This is a serious concern, and we need to identify specific requirements on the protocol (hopefully independent of particular scheduling/resource control schemes) to help address this.

Discovery: There needs to be some mechanism for a user to discover that there is a DECADE service available for their use, and to locate that server. This is particularly important in the case of mobile applications, since the actual servers that are available at any given time may differ. However, the specifics of what mechanisms (DHCP, HTTP page, etc.) have not been discussed, or even if the protocol should specify one or leave it as an implementation detail. This needs to be defined, and specific requirements formulated if needed.

Removal of Duplicate Records Across Servers: If the group wishes to allow for automated mechanisms to remove duplicates across a number of separate servers, some protocol support may need to be added. In the case of removing duplicates within a single (logical) DECADE server, this is simply an implementation concern. See Section 6 for more details.

8. Security Considerations

Authorization for access to in-network storage is an important part of the requirements listed in this document. Authorization for access to storage resources and the data itself is important for users to be able to manage and limit distribution of content. For example, a user may only wish to share particular content with certain peers.

If the authorization technique implemented in DECADE passes any private information (e.g., user passwords) over the wire, it MUST be passed in a secure way.

9. IANA Considerations

There are no IANA considerations with this document.

10. References

10.1. Normative References

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

10.2. Informative References

[I-D.ietf-decade-problem-statement]
Yongchao, S., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", draft-ietf-decade-problem-statement-00 (work in progress), August 2010.

[LLSB08] Dave Levin, Katrina LaCurts, Neil Spring, Bobby Bhattacharjee., "BitTorrent is an Auction: Analyzing and Improving BitTorrent's Incentives", In SIGCOMM 2008.

[PPLive] "PPLive", http://www.pplive.com.

Appendix A. Acknowledgments

We would also like to thank Haibin Song for substantial contributions to earlier versions of this document. We would also like to thank Reinaldo Penno, Alexey Melnikov, Rich Woundy, Ning Zong, Roni Even, David McDysan, Boerje Ohlman and Dirk Kutscher for contributions (including some text used verbatim) and general feedback.

Authors' Addresses

Yingjie Gu
Huawei
No. 101 Software Avenue
Nanjing, Jiangsu Province 210012
P.R.China

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

David A. Bryan
Cogent Force, LLC / Huawei

Email: dbryan@ethernot.org

Yang Richard Yang
Yale University

Email: yry@cs.yale.edu

Richard Alimi
Google

Email: ralimi@google.com

DECADE
Internet-Draft
Intended status: Informational
Expires: February 14, 2013

Y. Gu
Huawei
D. Bryan
Ethernot.org
Y. Yang
Yale University
P. Zhang
Tsinghua University/Yale University
R. Alimi
Google

August 13, 2012

DECADE Requirements
draft-ietf-decade-reqs-08

Abstract

The target of the DECoupled Application Data Enroute (DECADE) system is to provide an open and standard in-network storage system for applications, primarily P2P (peer-to-peer) applications, to store, retrieve and manage their data. This draft enumerates and explains requirements, not only for storage and retrieval, but also for data management, access control and resource control, that should be considered during the design and implementation of a DECADE-compatible system. These are requirements on the entire system; some of the requirements may eventually be implemented by an existing protocol with/without some extensions (e.g., a protocol used to read and write data from the storage system). The requirements in this document are intended to ensure that a DECADE-compatible system architecture includes all of the desired functionality for intended applications.

Requirements Language

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

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-

Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on February 14, 2013.

Table of Contents

1. Introduction

The object of the DECoupled Application Data Enroute (DECADE) system is to provide an open and standard in-network storage for content distribution applications, where data is typically broken into one or more chunks and then distributed. This may already include many types of applications including P2P applications, IPTV (Internet Protocol Television), and VoD (Video on Demand). (For a precise definition of the applications targeted in DECADE system, see the definition for Target Application in Section 2.) Instead of always transferring data directly from a source/owner client to a requesting client, the source/owner client can write to and manage its content on its in-network storage. The requesting client can get the address of the in-network storage pertaining to the source/owner client and read data from the storage.

This draft enumerates and explains the rationale behind specific requirements on the protocol design and on any data store implementation that may be used to implement DECADE servers that should be considered during the design and implementation of a DECADE-compatible system. As such, it does not include general guiding principles. General design considerations, explanation of the problem being addressed, and enumeration of the types of applications to which a DECADE-compatible system may be suited is not considered in this document. For general information, please see [RFC6646] and [I-D.ietf-decade-arch].

This document enumerates the requirements to enable target applications to utilize in-network storage. In this context, using storage resources includes not only basic capabilities such as writing, reading, and managing data, but also controlling access for particular remote clients with which it is sharing data. Additionally, we also consider controlling the resources used by remote clients when they access data as an integral part of utilizing the network storage.

This document discusses requirements pertaining to DECADE-compatible protocol(s). In certain deployments, several logical in-network storage systems could be deployed (e.g., within the same administrative domain). These in-network storage systems can communicate and transfer data through internal or non-standard communication messages that are outside of the scope of these requirements, but they should use DECADE-compatible protocol(s) when communicating with other DECADE-compatible in-network storage systems.

2. Terminology

This document uses the term 'In-network storage' which is defined in [RFC6646].

This document also defines these additional terms:

2.1. DECADE-compatible Client

A DECADE-compatible client uploads and/or retrieves data from DECADE-compatible servers. We use the shorter term "client" if there is no ambiguity.

2.2. DECADE-compatible Server

A DECADE-compatible server stores data inside the network, and thereafter manages both the stored data and access to those data. We use the shorter term "server" if there is no ambiguity.

2.3. DECADE Storage Provider

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

2.4. DECADE Account

An account of a DECADE-compatible server has associated cryptographic credentials for access control, and resource allocation attributes on the server.

2.5. Resource Provider

A client which has the account cryptographic credentials of a DECADE account at a DECADE-compatible server. We use the short term "Provider" if there is no ambiguity.

2.6. Resource Consumer

A client which tries to access a DECADE account but does not have the account's cryptographic credentials. We use the short term "Consumer" if there is no ambiguity.

2.7. Content Distribution Application

A Content Distribution Application is an application (e.g., P2P, traditional CDN, or hybrid P2P/CDN) designed for dissemination of a large amounts of content (e.g., files, or video streams) to multiple content consumers. Content Distribution Applications may divide

content into smaller blocks for dissemination.

2.8. Target Application

An application with that includes a DECADE-compatible client along with other application functionalities to explicitly control the usage of resources of DECADE-compatible servers to deliver content to other users. A primary type of Target Application we consider is Content Distribution Applications. A Target Application divides content into smaller blocks for more flexible distribution (e.g., over multiple application-level paths). We use the term Target Application to refer to the type of applications that are explicitly (but not exclusively) supported by DECADE system.

2.9. Application End-Point

An Application End-Point is an instance of a Target Application. A particular Application End-Point might be a Provider, a Consumer, or both. For example, an Application End-Point might be an instance of a video streaming client, or it might be the source providing the video to a set of clients.

2.10. Data Object

A data object is the unit of data stored and retrieved from a DECADE-compatible server. The data object is a string of raw bytes. The server maintains metadata associated with each data object, but the metadata is not included in the data object.

2.11. DECADE-compatible System

A system which is composed of DECADE-compatible clients, DECADE-compatible servers and in-network storage. A DECADE-compatible system MUST obey all the requirements defined in this document.

2.12. DECADE Resource Protocol (DRP) Functions

A set of functions for communication of access control and resource scheduling policies from a DECADE client to a server, as well as between servers.

2.13. DECADE Standard Data Transfer Protocol (SDT) Functions

A set of functions to be used to transfer data objects to and from a DECADE server.

3. System and Protocol Components

To organize requirements, we consider that a DECADE-compatible system consists of two logical sets of functions, as shown in Figure 1. The first set of functions, which we refer to as the DECADE Resource Protocol (DRP) functions, is responsible for communication of access control and resource scheduling policies from a client to a server, as well as between servers. A DECADE-compatible system will include exactly one DRP for interoperability and a common format through which these policies can be communicated.

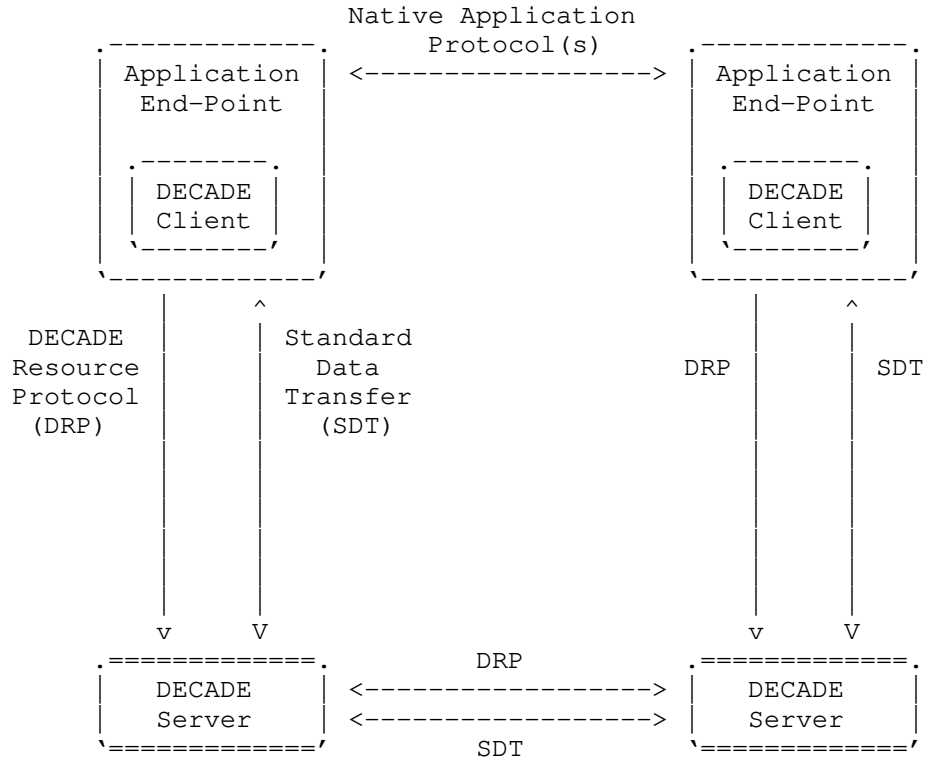


Figure 1: Protocol Components and Generic Flow

Second, the second set of functions, referred to as the Standard Data Transfer (SDT) functions, will be used to transfer data objects to and from a server. A DECADE-compatible system may support multiple SDT's. If there are multiple SDT's, a negotiation mechanism will be used to determine a suitable SDT between the client and server.

The two sets of functions (DRP and SDT) will be either separate or combined on the wire. If they are combined, DRP messages can be

piggy-backed within some extension fields provided by certain SDT protocols. In such a scenario, DRP is technically a data structure (transported by other protocols), but it can still be considered as a logical protocol that provides the services of configuring DECADE-compatible resource usage. If the protocols are physically separate on the wire, DRP can take the form of a separate control connection open between the a DECADE-compatible client and server. Hence, this document considers SDT and DRP as two separate, logical functional components for clarity.

4. Requirements Structure

This document specifies the requirements for the DECADE DRP and SDT functions, either existing ones or new ones, and storage system to enable Target Applications to make use of storage within the network, leaving specific storage system considerations to the implementation of the storage servers as much as possible. For this reason, we consider two primary categories of requirements:

- Protocol Requirements: Protocol requirements for Target Applications to make use of in-network storage within their own data dissemination schemes. Development of these requirements is guided by a study of data access, search and management capabilities used by Target Applications. These requirements may be met by a combination of existing protocols and new protocols.

- Storage Requirements: Functional requirements necessary for the back-end storage system employed by the DECADE server. Development of these requirements is guided by a study of the data access patterns used by Target Applications. These requirements should be met by the underlying data transport used by DECADE system. In this document, we use "data transport" to refer to a protocol used to read and write data from in-network storage.

This specification discusses the requirements of functionality implemented with a storage system and within applications, to permit interoperable communications concerning the manipulation of stored content.

5. Data Object Requirements

5.1. Data Name Uniqueness

REQUIREMENT(S): Each Data Object should be named to allow access. DECADE-compatible protocol(s) MUST support a data object naming scheme that ensures a high probability of uniqueness, with no coordination among multiple Storage Providers. In other words, two Data Objects with the same name should be the same content with high probability. A DECADE-compatible server SHOULD be able to respond to a DECADE-compatible client with an error indicating potential name conflicts.

RATIONALE: Although the intention of unique names is to avoid name collisions, it does not have to be an absolutely zero possibility. Hence, it is required to provide a collision handling mechanism.

EXCEPTION: While a DECADE-compatible server is overloaded or consider a request as an attack, it does not to generate a response to indicate name collisions.

5.2. Verifiable Name-Object Binding

5.3. Data Object Size

REQUIREMENT(S): DECADE MUST allow for efficient storage and data transfer of small data objects (e.g., 16KB) without large control overhead.

RATIONALE: Though Target Applications are frequently used to share large amounts of data (e.g., continuous streams or large files), the data itself is typically subdivided into smaller data objects (chunks) for flexibility (e.g., reliability and multi-path transmission).

5.4. Data Object Attributes

REQUIREMENT(S): DECADE MUST support the ability to associate a set of system attributes with a data object with a scope local to a DECADE-compatible server. In particular, the set MUST include time-to-live (or expiration time), creation timestamp, object size, and object type. A DECADE-compatible client, with access permission, MUST be able to query the set of system attributes. The transmission of the attributes MUST use an operating system-independent and architecture-independent standard format. An ability to extend the set of attributes MUST exist.

RATIONALE: The values of attributes associated with a data object are local to a particular DECADE-compatible server. These attributes may be used as hints to the storage system, internal optimizations, or as additional information query-able by DECADE-

compatible clients. The particular requirement for including time-to-live (TTL) is that a data object written by a DECADE-compatible client may be usable only within a certain window of time, such as in a live-streaming P2P application. Providing a time-to-live value for a data object (e.g., at the time it is written) can reduce management overhead by avoiding many 'delete' commands sent to DECADE-compatible server. The server may still retain a data object for bandwidth optimization, but this should be guided by the privacy policy of the DECADE Storage Provider.

5.5. Application-defined Object Properties and Metadata

REQUIREMENT(S): DECADE-compatible clients and DECADE-compatible servers MUST NOT be able to associate Application-defined properties (metadata) with data objects beyond what is provided by Section 5.4.

RATIONALE: Associating key-value pairs that are defined by Target Applications with data objects introduces substantial complexity. If Target Applications wish to associate additional metadata with a data object, possible alternatives include (1) managing such metadata within the Target Application itself, (2) storing metadata inside the data object, or (3) storing metadata in a different data object at the DECADE-compatible server.

6. Access and Authorization Requirements

6.1. Provider Access

REQUIREMENT(S): A Provider MUST be able to access the resources of its account.

RATIONALE: After a DECADE-compatible client owns an account on a DECADE-compatible server, it should be able to read data from and write data to the server.

6.2. Secure Authorization

REQUIREMENT(S): Access to an account on a server MUST be granted to a provider based on cryptographic security.

RATIONALE: DECADE-compatible clients may be operating on hosts without constant network connectivity or without a permanent attachment address (e.g., mobile devices). To support access control with such hosts, DECADE-compatible servers must support access control policies that use cryptographic credentials, not simply by tying access to IP addresses.

6.3. Consumer Access

REQUIREMENT(S): A Provider MUST be able to indicate to its server on whether a Consumer can access its subscribed resources.

RATIONALE: Endpoints in Target Applications may choose different servers. Thus, to be useful by Target Applications, a DECADE-compatible client must be able to specify policies on whether other DECADE-compatible clients can access its resources. The other clients may or may not be known to the server.

6.4. Provider Authorization When Offline

REQUIREMENT(S): A Provider MUST be able to grant access to a Consumer even if the Provider is not actively running or connected to its DECADE-compatible server.

RATIONALE: If an application desires, it can authorize other clients to access its storage even after the application exits or network connectivity is lost. An example use case is mobile scenarios, where a client can lose and regain network connectivity often.

6.5. Local Authorization

REQUIREMENT(S): A Provider MUST be able to indicate, without contacting its server, access control policies for Consumers. DECADE-compatible server MUST be able to authenticate the access control policies in this situation.

RATIONALE: This requirement is related with the preceding requirement, but in a perspective (i.e., protocol design). See discussions in Section 8.3.

6.6. Access Control Granularity

REQUIREMENT(S): A Provider MUST be able to control which individual clients are authorized to read/write which particular data objects from/to its in-network storage.

RATIONALE: A Target Application should able to conduct access control on the granularity of individual clients, individual data objects.

6.7. Default Access Permissions

REQUIREMENT(S): Unless read or write access is granted by a Provider, the default permission MUST be no access.

RATIONALE: This requirement is to protect client privacy by default.

6.8. Connectivity Supporting NAT and Firewall Traversal

REQUIREMENT(S): A client that is authorized to access a server MUST be supported to conduct NAT (Network Address Translation) and firewall traversal. In particular, network connections between a DECADE-compatible client and a DECADE-compatible server MUST be initiated by the client (i.e., the server must not initiate a connection to the client).

RATIONALE: Firewalls and NATs are widely used in the Internet today, both in ISP (Internet Service Provider) and Enterprise networks and by consumers. Many firewalls and NATs are configured by default to block incoming connections, which helps to mitigate security risks. Deployment of a DECADE-compatible system must not require manual modifications to such devices. This requirement applies to both potential new protocol that may be developed by the DECADE Working Group and any data transport used with DECADE protocol.

6.9. DECADE Server Discovery

REQUIREMENT(S): A mechanism for a Provider to discover and connect to its assigned server MUST be supported. The discovery SHOULD leverage existing mechanisms and protocols wherever possible. No new discovery mechanism will be defined unless there is enough evidence that no existing mechanism can work.

RATIONALE: Existing protocols such as DNS and DHCP are widespread. Using existing protocols, or combinations of the protocols that have been specified in other contexts, is strictly preferred over developing a new discovery protocol.

7. Data Transfer Requirements

7.1. Negotiable Standard Data Transport Protocol

REQUIREMENT(S): A DECADE-compatible client MUST be able to negotiate with a DECADE-compatible server about which protocol it can use to read data from and write data. DECADE MUST specify at least one mandatory transport protocol to be supported by implementations; usage of a different protocol may be selected via negotiation.

RATIONALE: Since typical data transport protocols (e.g., NFS and WebDAV) already provide read and write operations for network storage, it may not be necessary to define such operations in a new DECADE protocol. However, because of the particular application requirements and deployment considerations, different applications may support different protocols. Thus, a DECADE client must be able to select an appropriate protocol also supported by the in-network storage.

7.2. Atomic or Partial Read/Write

REQUIREMENT(S): A DECADE-compatible server MUST support the ability to read/write a complete data object in one request. It MAY support range read/write.

RATIONALE: Depending on the object size (e.g., chunk size) used by a Target Application, the application may need to send data to the DECADE-compatible server in multiple round.

7.3. Secure Data Transport

REQUIREMENT(S): A secure transport MUST be implemented for all communications between a DECADE-compatible client and DECADE-compatible server.

RATIONALE: Target Applications may wish to write sensitive data. To satisfy this use case, the communication between a DECADE-compatible client and DECADE-compatible server must be transported over a secure transport protocol (e.g., SSL/TLS).

7.4. Concurrent Read

REQUIREMENT(S): A DECADE-compatible server MUST allow for multiple simultaneous readers for a data object.

RATIONALE: One characteristic of Target Applications is the ability to upload an object to multiple clients. Thus, it is natural for DECADE-compatible server to allow multiple readers to read the same object concurrently.

7.5. Concurrent Write

REQUIREMENT(S): A DECADE-compatible server MUST NOT allow multiple simultaneous writers for the same object. A DECADE-compatible server SHOULD respond to each of the writers with an error.

RATIONALE: This avoids data corruption in a simple way while remaining efficient. Alternately, the DECADE-compatible server would need to either manage locking, handle write collisions, or merge data, all of which reduce efficiency and increase complexity.

EXCEPTION: While a DECADE-compatible server is overloaded or considers a request as an attack, it does not generate a response.

7.6. Read Before Write Complete

REQUIREMENT(S): A DECADE-compatible server MAY allow readers to read a data object before it has been completely written. In case of a write error in such a case, the DECADE-compatible server SHOULD respond to the reading client with an error indicating that the write has failed.

RATIONALE: Some Target Applications (in particular, P2P streaming) can be sensitive to latency. A technique to reduce latency is to remove store-and-forward delays for data objects (e.g., make the object available before it is completely written). Appropriate handling for error conditions (e.g., a disappearing writer) needs to be specified.

EXCEPTION: While a DECADE-compatible server is overloaded or considers a request as an attack, it does not generate a response.

7.7. Redirection of Transport

REQUIREMENT(S): A DECADE-compatible server SHOULD be able to redirect requests to another DECADE-compatible server. This may either be in response to an error, failure, overload, or to support capabilities such as load balancing.

RATIONALE: A DECADE-compatible server may opt to redirect requests to another server to support capabilities such as load balancing, or if the implementation decides that another DECADE-compatible server is in a better position to handle the request due to either its location in the network, server status, or other consideration.

EXCEPTION: A DECADE-compatible server can be configured by its service provider to support or not support redirection functionality.

8. Resource Control Requirements

8.1. Multiple Applications Sharing Resources

REQUIREMENT(S): A client MUST be able to indicate to a DECADE-compatible server about resource sharing policies among multiple Target Applications being run/managed by the same client.

RATIONALE: A client owning a DECADE account may provide the account's cryptographic credentials to multiple Providers of multiple target applications. For example, the client may run one or more video-on-demand application(s) and a live-streaming application(s) which both make use of the client's in-network storage. The concurrently running applications may be running on different machines (e.g., multiple machines at the same home network) and may not directly communicate, except through user coordination

8.2. Per-Client Resource Policy

REQUIREMENT(S): A Provider MUST be able to specify resource policies (bandwidth share, storage quota, and network connection quota) to individual Consumers for reading from and writing data to its in-network storage within a particular range of time.

RATIONALE: Target Applications can rely on control of resources on a per-client basis. For example, application policy may indicate that certain remote clients have a higher bandwidth share for receiving data [LLSB08]. Additionally, bandwidth share for receiving data [LLSB08]. Additionally, certain data (e.g., chunks) may be distributed with a higher priority. As another example, when allowing a remote client to write data to a user's in-network storage, the remote client may be restricted to write less than 100MB of data in total. Since the client may need to manage multiple clients accessing its data, it should be able to indicate the time over which the granted resources are usable. For example, an expiration time for the access could be indicated to the DECADE-compatible server after which no resources are granted (e.g., indicate error as access denied).

8.3. Distributed Resource Sharing Specification

REQUIREMENT(S): A Provider MUST be able to indicate to its DECADE-compatible server, without itself contacting the server, resource control policies for a Consumer. The DECADE-compatible server MUST be able to authenticate the resource control policies.

RATIONALE: One important consideration for a DECADE-compatible server is scalability, since a single storage element may be used to support many users. Many Target Applications use small chunk sizes and frequent data exchanges. If such an application employed resource control and contacted the DECADE-compatible server for each data exchange, this could present a scalability challenge for the server as well as additional latency for clients.

The preferred way is to let requesting clients obtain signed resource control policies (in the form of a token) from the owning client, and then requesting clients can present the policy to a DECADE-compatible server directly. This can result in reduced messaging handled by the DECADE-compatible server.

8.4. Resource Set

REQUIREMENT(S): A DECADE-compatible server MUST allow specification on the following resources: storage, bandwidth, and number of connections, and MAY allow additional types of resources to be specified.

RATIONALE: The minimum set of resources need to include the most common resources.

9. Error and Failure Handling Requirements

9.1. Illegal Data Object

REQUIREMENT(S): A DECADE-compatible server SHOULD provide an error indicating that (1) data was rejected from being written, (2) deleted, or (3) marked unavailable.

RATIONALE: DECADE Storage Providers may require the ability to (1) avoid storing, (2) delete, or (3) quarantine certain data that has been identified as illegal (or otherwise prohibited). It is not specified how such data is identified, but applications employing DECADE-compatible servers should not break if a Storage Provider is obligated to enforce such policies. Appropriate error conditions should be indicated to applications.

EXCEPTION: A DECADE-compatible server should be able to be configured to suppress Illegal Data Object responses for security reasons.

9.2. Invalid Access Authorization

REQUIREMENT(S): A DECADE-compatible server SHOULD provide an error indicating that the request does not have a valid access authorization.

RATIONALE: DECADE-compatible clients may request data objects to which they do not have a valid authorization, and DECADE-compatible servers should be able to signal that this has occurred. Invalid authorization may be due to a combination of credential issues as well as additional policies defined by a Storage Provider.

EXCEPTION: A DECADE-compatible server should be able to be configured to suppress Invalid Authorization responses for security reasons.

9.3. Insufficient Resources

REQUIREMENT(S): A DECADE-compatible server SHOULD provide a response indicating to a DECADE-compatible client that resources (e.g., storage space) were not available to service its request (e.g., storage quota exceeded when attempting to write data).

RATIONALE: The Insufficient Resources response allows a client to back off, free up necessary resources or waiting for such resources to be freed.

EXCEPTION: A DECADE-compatible server may not provide such a response if doing so increases the load or due to security concerns.

9.4. Overload Condition

REQUIREMENT(S): A DECADE-compatible server, which is operating close to its capacity limit (e.g., too busy servicing other requests), MUST be permitted to reject requests and not be required to generate response to additional requests. A DECADE-compatible server MUST also be permitted to redirect requests as a load-shedding technique.

RATIONALE: The Insufficient Resources response allows a client to back off, free up necessary resources or waiting for such resources to be freed.

EXCEPTION: A DECADE-compatible server may not provide such a response if doing so increases the load or due to security concerns.

9.5. Attack Mitigation

REQUIREMENT(S): A DECADE-compatible server MUST be permitted to reject suspicious requests and not be required to generate responses (e.g., if a client's rate of requests exceeds a pre-defined threshold).

RATIONALE: Malicious clients may attempt to attack a DECADE-compatible server by specifying many chunks to increase total throughput or inciting overload conditions. A DECADE-compatible server is permitted to reject or ignore requests that are deemed suspicious according to policies set by its DECADE service provider.

10. Management Info Requirements

10.1. Account Status

REQUIREMENT(S): A Provider MUST be able to query the resource quota as well the current usage. The response from the server MUST include resource usage resulting from both the client's own usage and usage by other clients that have been authorized to read/write objects on that client's account.

RATIONALE: The resources used by a client are not necessarily directly-attached (e.g., disk, network interface, etc). Thus, the client cannot locally determine how much resources are being used. Before storing and retrieving data, a client should be able to determine which data is available (e.g., after an application restart).

11. Security Considerations

The security model is an important component of a DECADE-compatible system. It is crucial for users to be able to manage and limit distribution of content to only authorized parties, and the mechanism needs to work on the general Internet which spans multiple administrative and security domains. Previous sections have enumerated detailed requirements, but this section discusses the overall approach and other considerations that do not merit requirements.

11.1. Authentication and Authorization

A DECADE-compatible server must validate an request to access the in-network storage.

11.2. Confidentiality

DECADE-compatible Servers provide the ability to write raw data objects (subject to any policies instituted by the owner/ administrator of the Storage Provider). Thus, DECADE-compatible clients may opt to encrypt data before it is transported to the server.

11.3. Attack Mitigation

The particular resource control policy may be open to certain attacks by clients. For example by specifying many small chunks to increase total throughput or inciting overload conditions are techniques that may be used by a client.

12. IANA Considerations

There are no IANA considerations with this document.

13. References

13.1. Normative References

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

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

13.2. Informative References

[I-D.ietf-decade-arch] Alimi, R., Rahman, A., Kutscher, D., and Y. Yang, "DECADE Architecture", draft-ietf-decade-arch-08 (work in progress), July 2012.

[LLSB08] Levin, D., LaCurts, K., Spring, N., and B. Bhattacharjee, "BitTorrent is an Auction: Analyzing and Improving BitTorrent's Incentives", SIGCOMM 2008, August 2008.

Appendix A. Acknowledgments

We would also like to thank Haibin Song for substantial contributions to earlier versions of this document. We would also like to thank Reinaldo Penno, Alexey Melnikov, Rich Woundy, Ning Zong, Roni Even, David McDysan, Borje Ohlman, Dirk Kutscher, Akbar Rahman, Xiao Zhu, Yunfei Zhang, Peng Zhang and Jin Peng for contributions and general feedback.

Authors' Addresses

Yingjie Gu
Huawei
No. 101 Software Avenue
Nanjing, Jiangsu Province 210012
P.R.China

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

David A. Bryan
Ethernot.org

Email: dbryan@ethernot.org

Yang Richard Yang
Yale University

Email: yry@cs.yale.edu

Peng Zhang
Tsinghua University/Yale University

Email: p.zhang@yale.edu

Richard Alimi
Google

Email: ralimi@google.com

DECADE
Internet-Draft
Intended status: Informational
Expires: September 3, 2011

R. Alimi, Ed.
Google
A. Rahman, Ed.
InterDigital Communications, LLC
Y. Yang, Ed.
Yale University

March 2, 2011

A Survey of In-network Storage Systems
draft-ietf-decade-survey-04

Abstract

This document surveys deployed and experimental in-network storage systems and describes their applicability for DECADE.

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 3, 2011.

Table of Contents

1. Introduction

DECADE (DECoupled Application Data Enroute) is an architecture that provides applications with access to in-network storage. With access to in-network storage, content distribution applications can be designed to place less load on network infrastructure, such as last-mile links. See [1] for further discussion.

A major motivation for DECADE is the substantial increase of capacity and reduction in cost offered by storage systems. For example, over the last two decades, there has been at least a 30-fold increase in the amount of storage that you can get for a given price (for flash memory and hard disk drives) [2], [3].

High-capacity and low-cost in-network storage devices introduces substantial opportunities. One example of in-network storage is content caches supporting Web and Peer-to-Peer (P2P) content. Different from existing content caches whose control fully reside at the owners of the caching devices, DECADE also allows applications to control access to their allocated in-network storage, as well as the resources consumed while accessing that storage (bandwidth, connections, storage space). While designed in the context of P2P applications, it may be useful to other applications as well. This document provides details on deployed and experimental in-network storage solutions, and evaluates their suitability for DECADE.

We note that the survey presented in this document is only representative of the research in this area. Rather than trying to enumerate an exhaustive list, we have chosen some typical techniques that lead to derivative works.

2. Survey Overview

2.1. Terminology and Concepts

This document uses terms defined in [1].

2.2. Historical Context

In-network storage has been used previously in numerous scenarios to reduce network traffic and enable more efficient content distribution. This section presents a brief history of content distribution techniques and illustrates how DECADE relates to past approaches. Systems have been developed with particular use cases in mind. Thus, this survey is not meant to point out shortcomings of existing solutions, but rather to indicate where certain capabilities required in DECADE [4] are not provided by existing systems.

In the early stage of Internet development, most Web content was stored at a central server and clients requested Web content from the central server. In this architecture, the central server was required to provide a large amount of bandwidth. Web browsing is still a primary activity on today's Internet. As more and more users access Web content, a central server can become overloaded. The use of web caches is one technique to reduce load on a central server. Web caches store frequently-requested content, and provide bandwidth for serving the content to clients.

The ongoing growth of broadband technology in the worldwide market has been driven by the hunger of customers for new multimedia services as well as Web content. In particular, the use of audio and video streaming formats has become common for delivery of rich information to the public - both residential and business.

To overcome this challenge of massive multimedia consumption, just installing more Web cache will not be enough. Moving content closer to the consumer results in greater network efficiency, improved QoS, and lower latency, while facilitating personalization of content through broadband content applications. In these edge technologies, CDN is a representative technique. Content Delivery Networks (CDN) are based on a large-scale distributed network of servers located closer to the edges of the Internet for efficient delivery of digital content including various forms of multimedia content.

Although CDN is an effective means of information access and delivery, there are two barriers to making CDN a more common service: cost and replication integrity. Deploying a CDN for publicly available content is expensive. It requires administrative control over nodes with large storage capacity at geographically dispersed locations with adequate connectivity. CDN can be scalable, but due to this administrative and cost overhead, not rapidly deployable for the common user.

The emergence and maturation of P2P has allowed improvements to many network applications. P2P allows the use of client resources, such as CPU, memory, storage, and bandwidth, for serving content. This can reduce the amount of resources required by a content provider. Multimedia content delivery using various P2P or peer-assisted frameworks has been shown to greatly reduce the dependence on CDN and central content servers. However, the popularity of P2P applications has resulted in increased traffic on ISP networks. P2P caches (both transparent and non-transparent) have been introduced as a way to reduce the burden. Though they can be effective in reducing traffic in certain areas of ISP networks, P2P caches have their shortcomings. In particular, they are application-dependent and thus difficult to keep up-to-date with new and evolving P2P application protocols.

Second, applications may benefit from explicit control of in-network storage, which P2P caches do not provided. See [1] for further discussion.

DECADE aims to provide a standard protocol allowing P2P applications (including Content Providers) to make use of in-network storage to reduce the traffic burden on ISP networks, while enabling P2P applications to control access to content they have placed in in-network storage.

3. In-network Storage System Components

Before surveying individual technologies, we describe the basic components of in-network storage. For consistency and for ease of comparison, we use the same model to evaluate each storage technology in this document.

Note that the network protocol(s) used by a given storage system are also an important part of the design. We omit details of particular protocol choices in this document.

3.1. Data Access Interface

A set of operations available to a client user for accessing data in the in-network storage. Solutions typically allow both read and write, though the mechanisms for doing so can differ drastically.

3.2. Data Management Operations

Storage systems may provide users the ability to manage stored content. For example, operations such as delete and move may be provided to users. In this survey, we focus on data management operations that are provided to client users and omit those provided to system administrators.

3.3. Data Search Capability

Some storage systems may provide the capability to search or enumerate content that has been stored. In this survey, we focus on search capabilities that are provided to client users and omit those provided to system administrators. An example of a client search would be to find out the list of items stored by the given user over a given period of time.

3.4. Access Control Authorization

Storage systems typically allow a client user, content owner or some other entity to define the access policies for the in-network storage. The in-network storage system then checks the authorization of a user before it stores or retrieves content. We define three types of access control authorization: public-unrestricted, public-restricted, and private.

Public-unrestricted refers to content on an in-network storage system that is widely available to all clients (i.e., without restrictions). An example is accessing Wikipedia on the Web, or anonymous access to FTP sites.

Public-restricted refers to content on an in-network storage system that is available to a restricted (though still potentially large) set of clients, but which do not require any confidential credentials from the client. An example is some content (e.g., a TV show episode) on the Internet that can only be viewable in selected countries or networks (i.e., white/black lists or black-out areas).

Private refers to content on an in-network storage system that is only made available to one or more clients presenting the required confidential credentials (e.g., password or key). This content is not available to anyone without the proper confidential access credentials.

Note that a combination of access control types may be applicable for a given scenario. For example, the retrieval (read) of content from an in-network storage system may be public-unrestricted, but the storage (write) to the same system may be private.

3.5. Resource Control Interface

This is the interface through which users manage the resources on in-network storage that can be used by other peers, e.g., the bandwidth or connections. The storage system may also allow users to indicate a time for which resources are granted.

3.6. Discovery Mechanism

Users use the discovery mechanism to find location of in-network storage, find access interface or resource control interface or other interfaces of in-network storage.

3.7. Storage Mode

Storage systems may use the following modes of storage: file system, object-based, or block-based.

A file system typically organizes files into a hierarchical tree structure. Each level of the hierarchy normally contains one or more directories each with one or more files. A file system may also be flat or use some other organizing principle.

We define an object-based storage mode as one which stores discrete chunks of data (e.g., IP datagrams or another type of aggregation useful to an application) without a pre-defined hierarchy or meta structure.

We define a block-based storage mode as one which stores a raw sequence of bytes, with a client being able to read and/or write data at offsets within that sequence. Data is typically accessed in blocks for efficiency. An common example for this storage mode is raw access to a hard disk.

In this survey, we define Storage Mode to refer to how data is structured within the system, which may not be the same as how it is accessed by a client. For example, a caching system may cache objects with hierarchical names, but may internally use an object-based Storage Mode.

4. In-Network Storage Systems

This section surveys in-network storage systems using the methodology defined above. The survey includes some systems that are widely deployed today, some systems that are just being deployed, and some experimental/futuristic systems. The survey covers both traditional client-server architectures and P2P architectures. The surveyed systems are listed in alphabetical order. Also, for each system, a brief explanation is given of the relevance to DECADE.

4.1. Amazon S3

Amazon S3 (Simple Storage Service) [5] provides an online storage service using web (HTTP) interfaces. Users create buckets, and each bucket can contain stored objects. Users are provided an interface through which they can manage their buckets. Amazon S3 is a popular backend storage for other services. Other related storage services is the Blob Service provided by Windows Azure [6], and the Google Storage for Developers [7].

4.1.1. Applicability to DECADE

Very widely used (deployed) example of in-network storage. Amazon leases the storage to third party companies for disparate services. In particular, Amazon S3 has a rich model for authorization (using signed queries) to integrate with a wide variety of use cases. A focus for Amazon S3 is scalability. Particular simplifications that were made are the absence of a general, hierarchical namespace and the inability to update the contents of existing data.

4.1.2. Data Access Interface

Users can read, and write objects.

4.1.3. Data Management Operations

Users can delete previously-stored objects.

4.1.4. Data Search Capability

Users can list contents of buckets to find objects matching desired criteria.

4.1.5. Access Control Authorization

All methods of access control are supported for clients: public-unrestricted, public-restricted and private.

For example, access to stored objects can be restricted by owner, a list of other Amazon Web Service users, all Amazon Web Service Users, or open to all users (anonymous access). Another option is for the owner to generate and sign a query (e.g., a query to read an object) that can be used by any user until an owner-defined expiration time.

4.1.6. Resource Control Interface

Not provided.

4.1.7. Discovery Mechanism

Users are provided a well-known DNS name (either a default provided by Amazon, or one customized by a particular user). Users accessing S3 storage use DNS to discover an IP address where S3 requests can be sent.

4.1.8. Storage Mode

Object-based, with the extension that objects can be organized into user-defined buckets.

4.2. BranchCache

BranchCache [8] is a feature integrated into Windows (Windows 7 and Windows Server 2008R2) that aims to optimize enterprise branch office file access over the WAN links. The main goals are to reduce WAN link utilization and improve application responsiveness by caching and sharing content within a branch while still maintaining end-to-end security. BranchCache allows files retrieved from the web servers and file servers located in headquarters or datacenters to be cached in remote branch offices, and shared among users in the same branch accessing the same content. BranchCache operates transparently by instrumenting the HTTP and SMB components of the networking stack. It provides two modes of operation: Distributed Cache and Hosted Cache.

In both modes, a client always contacts a BranchCache-enabled content server first to get the content identifiers for local search. If the content is cached locally, the client then retrieves the content within the branch. Otherwise, the client will go back to the original content server to request the content. The two modes differ in how the content is shared.

In the Hosted Cache mode, a locally provisioned server acts as a cache for files retrieved from the servers. After getting the content identifiers, the client first consults the cache for the desired file. If it is not present in the cache, the client retrieves it from the content server and sends it to the cache for storage.

In the Distributed Cache mode, a client first queries other clients in the same network using the Web Services Discovery multicast protocol. As in the Hosted Cache mode, the client retrieves the file from the content server if it is not available locally. After retrieving the file (either from another client or the content server), the client stores the file locally.

The original content server always authorizes requests from clients. Cached content is encrypted, and clients can only decrypt the data using keys derived from metadata returned by the content server. In addition to instrumenting the networking stack at clients, content servers must also support BranchCache.

4.2.1. Applicability to DECADE

BranchCache is an example of an in-network storage system primarily targeted at enterprise networks. It supports both a P2P like mode (Distributed Cache) as well as a client-server mode (Hosted Cache). Integration into the Microsoft OS will ensure wide distribution of this in-network storage technology.

4.2.2. Data Access Interface

Clients transparently retrieve (read) data from a cache (other clients or a Hosted Cache) since it operates by instrumenting the networking stack. In Hosted Cache mode, clients write data to the Hosted Cache once it is retrieved from the content server.

4.2.3. Data Management Operations

Not provided.

4.2.4. Data Search Capability

Not provided.

4.2.5. Access Control Authorization

Access control method for clients is private. For example, transferred content is encrypted, and can only be decrypted by keys derived from data received from the original content server. Though data may be transferred to unauthorized clients, end-to-end security is maintained by only allowing authorized clients to decrypt the data.

4.2.6. Resource Control Interface

The storage capacity of caches on the clients and Hosted Caches are configurable by system administrators. The Hosted Cache further allows configuration of the maximum number of simultaneous client accesses. In the Distributed Caching mode, exponential back-off and throttling mechanisms are utilized to prevent reply storms of popular content requests. The client will also spread data block access among multiple serving clients that have the content (complete or partial) to improve latency and provide some load balancing.

4.2.7. Discovery Mechanism

The Distributed Cache mode uses multicast for discovery of other clients and content within a local network. Currently, the Hosted Cache mode uses policy provisioning or manual configuration of the

server used as the Hosted Cache. In this mode, the address of the server may be found via DNS.

4.2.8. Storage Mode

Object-based.

4.3. Cache-and-Forward Architecture

Cache-and-Forward (CNF) [9] is an architecture for content delivery services for the future Internet. In this architecture, storage can be exploited at nodes within the network, either directly at routers or deployed nearby the routers. CNF is based on the concept of store-and-forward routers with large storage, providing for opportunistic delivery to occasionally disconnected mobile users and for in-network caching of content. The proposed CNF protocol uses reliable hop-by-hop transfer of large data files between CNF routers in place of an end-to-end transport protocol like TCP.

4.3.1. Applicability to DECADE

An example of an experimental in-network storage system that would require storage space on (or near) a large number of routers in the Internet if it was deployed. As the name of the system implies, it would provide short term caching and not long term network storage.

4.3.2. Data Access Interface

Users implicitly store content at Cache-and-forward routers by requesting files. End hosts read content from in-network storage by submitting queries for content.

4.3.3. Data Management Operations

Not provided.

4.3.4. Data Search Capability

Not provided.

4.3.5. Access Control Authorization

Access control method is public-restricted (to any client which is part of the cache-and-forward network).

4.3.6. Resource Control Interface

Not provided.

4.3.7. Discovery Mechanism

A query including a location-independent content ID is sent to the network, and routed to a Cache-and-forward router, which handles retrieval of the data and forwarding to the end host.

4.3.8. Storage Mode

Object-based (with objects representing individual files). The architecture proposes to cache large files in storage within the network, though objects could be made to represent smaller chunks of larger files.

4.4. Content Delivery Network

A Content Delivery Network (CDN) provides services that improve network performance by maximizing bandwidth, improving accessibility and maintaining correctness through content replication. They offer fast and reliable applications and services by distributing content to cache or edge servers located close to users. See [10] for an additional taxonomy and survey.

A CDN has some combination of content-delivery, request-routing, distribution and accounting infrastructure. The content-delivery infrastructure consists of a set of edge servers (also called surrogates) that deliver copies of content to end-users. The request-routing infrastructure is responsible for directing client requests to appropriate edge servers. It also interacts with the distribution infrastructure to keep an up-to-date view of the content stored in the CDN caches. The distribution infrastructure moves content from the origin server to the CDN edge servers and ensures consistency of content in the caches. The accounting infrastructure maintains logs of client accesses and records the usage of the CDN servers. This information is used for traffic reporting and usage-based billing.

In practice, a CDN typically hosts static content including images, video, media clips, advertisements, and other embedded objects for Web viewing. A focus for CDNs is the ability to publish and deliver content to end-users in a reliable and timely manner. A CDN focuses on building its network infrastructure to provide the following services and functionalities: storage and management of content; distribution of content among surrogates; cache management; delivery of static, dynamic and streaming content; backup and disaster

recovery solutions; and monitoring, performance measurement and reporting.

Examples of existing CDNs are Akamai, Limelight, and CloudFront.

The following description uses the term "content provider" to refer to the entity purchasing a CDN service, and the term "client" to refer to the subscriber requesting content via the CDN from the content provider.

4.4.1. Applicability to DECADE

Very widely used (deployed) example of in-network storage for multimedia content. The existence and operation of the storage is totally transparent to the end user. A CDN typically require a strong business relationship between the content providers and content distributors and often the business relationship extends to the ISPs.

4.4.2. Data Access Interface

A CDN is typically a closed system, and generally provides only read (retrieve) access interface to clients. A CDN typically does not provide write (store) access interface to clients. The content provider can access network edge servers and store content on them. Or edge servers can retrieve content from content providers. Client nodes can just retrieve content from edge servers.

4.4.3. Data Management Operations

A content provider can manage the data distributed in different cache nodes, such as moving popular data objects from one cache node to another cache node, or deleting rarely-accessed data objects in cache nodes. Client user nodes, however, have no right to perform these operations.

4.4.4. Data Search Capability

A content provider can search or enumerate the data each cache node stores. Client user nodes cannot perform search operations.

4.4.5. Access Control Authorization

All methods of access control (for reading) are supported for clients: public-unrestricted, public-restricted and private. Some CDN edge servers will allow usage of HTTP basic authentication with the origin server, restrictions by IP address, or they can use a token-based technique to allow the origin server to apply its own

authorization criteria.

Also as mentioned previously, clients typically cannot write to the CDN. Writing is typically a private operation for the content providers.

4.4.6. Resource Control Interface

Not provided.

4.4.7. Discovery Mechanism

Content providers can directly find internal CDN cache nodes to store content, since they typically have an explicit business relationship. Clients can locate CDN nodes through DNS or other redirection mechanism.

4.4.8. Storage Mode

Though addressing objects uses URLs which typically refer to objects in a hierarchical fashion, the storage mode is typically object-based.

4.5. Delay-Tolerant Network

The Delay-Tolerant Network (DTN) [11] is an evolution of an architecture originally designed for the Interplanetary Internet. The Interplanetary Internet is a communication system envisioned to provide Internet-like services across interplanetary distances in support of deep space exploration. The DTN architecture can be utilized in various operational environments characterized by severe communication disruptions, disconnections and high-delays (e.g., a month long loss of connectivity between two planetary networks because of high solar radiation due to sun spots). The DTN architecture is thus suitable for environments including deep space networks, sensor-based networks, certain satellite networks and underwater acoustic networks.

A key aspect of the DTN is a store and forward overlay layer called the "Bundle Protocol" or "Bundle Layer" that exists between the transport and application layers [12]. The Bundle Layer forms a logical overlay that employs persistent storage to help combat long term network interruptions by providing a store and forwarding service. While traditional IP networks are also based on store and forward principles, the amount of time of a packet being kept in "storage" at a traditional IP router is typically in the order of milli-seconds (or less). In contrast, the DTN architecture assumes that most Bundle Layer nodes will use some form of persistent storage

(e.g., hard disk, flash memory, etc.) for DTN packets because of the nature of the DTN environment.

4.5.1. Applicability to DECADE

An example of an experimental in-network storage system that would require fundamental changes to the Internet protocols.

4.5.2. Data Access Interface

Users implicitly cause content to be stored (until successfully forwarded) at Bundle Layer nodes by initiating/terminating any transaction that traverses the DTN.

4.5.3. Data Management Operations

Users can implicitly cause deletion of content stored at Bundle Layer nodes via a "Time To Live" type parameter that the user can control (for transactions originating from the user).

4.5.4. Data Search Capability

Not provided.

4.5.5. Access Control Authorization

Access control method is public-restricted (to any client which is part of the DTN) or private.

4.5.6. Resource Control Interface

Not provided.

4.5.7. Discovery Mechanism

A Uniform Resource Identifier (URI) approach is used as the basis of the addressing scheme for DTN transactions (and subsequent store and forward routing through the DTN network).

4.5.8. Storage Mode

Object-based. DTN applications send data to the Bundle Layer which then breaks the data into segments. These segments are then routed through the DTN network, and stored in Bundle Layer nodes as required (before being forwarded).

4.6. Named Data Networking

Named Data Networking (NDN) [13] is a research initiative which proposes to move to a new model of addressing and routing for the Internet. NDN uses "named data" based routing and forwarding, to replace the current IP address based model. NDN also uses name-based data caching in the routers.

Each NDN Data packet will be assigned a content name and will be cryptographically signed. Data delivery is driven by the requesting end. Routers disseminate name-based prefix announcements by using routing protocols like Intermediate System to Intermediate System (IS-IS) or Border Gateway Protocol (BGP). The requester will send out an "Interest" packet which identifies the name of the data that it wants. Routers that receive this Interest packet will remember the interface it came from and will then forward it on a named-based routing protocol. Once an Interest packet reaches a node that has the desired data, a named Data packet is sent back, which carries both the name and content of the data, along with a digital signature of the producer. This named Data packet is then forwarded back to the original requester on the reverse path of the Interest packet [14].

A key aspect of NDN is that router have the capability to cache the named data. If a request for the same data (i.e., same name) comes to the router, then the NDN router will forward the named data stored locally to fulfill the request. The proponents of NDN believe that the network can be designed naturally matching data delivery characteristics instead of communication between endpoints because data delivery has become the primary use of the network.

4.6.1. Applicability to DECADE

An example of an experimental in-network storage system that would require storage space on a large number of routers in the Internet. Named Data packets would be kept in storage in the NDN routers and provided to new requesters of the same data.

4.6.2. Data Access Interface

Users implicitly store content at NDN routers by requesting content (named Data packets) from the network. Subsequent requests by different users for the same content will cause the named Data packets to be read from the NDN routers in-network storage.

4.6.3. Data Management Operations

Users do not have the direct ability to delete content stored in the NDN routers. However, there will be some type of "Time To Live" parameter associated with the named Data packets though this has not yet been specified.

4.6.4. Data Search Capability

Not provided.

4.6.5. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private.

The basic security mechanism in NDN is for the sender to digitally sign the content (named Data packet) that it sends. It is envisioned that a complete access control system can be built on top of this though this has not yet been specified.

4.6.6. Resource Control Interface

Not provided.

4.6.7. Discovery Mechanism

Names are used as the basis of the addressing and discovery scheme for NDN (and subsequent store and forward routing through the NDN network). NDN names are assumed to be hierarchical and to be able to be deterministically constructed. This is still an active area of research.

4.6.8. Storage Mode

Object-based. NDN sends named Data packets through the network. These Data packets are routed through the NDN network, and stored in NDN routers.

4.7. Network of Information

Similar to NDN (see Section 4.6), Network of Information (NetInf) [15] is another information centric approach in which the named data objects are the basic component of the networking architecture. NetInf is thus moving away from today's host centric networking architecture where the nodes in the network are the primary objects. In today's network the information objects are named relative to the hosts they are stored on (e.g.,

http://www.example.com/information-object.txt).

The NetInf naming and security framework builds the foundation for an information centric security model that integrates security deeply into the architecture. In this model, trust is based on the information itself. Information Objects (IOs) are given a unique name with cryptographic properties. Together with additional metadata, the name can be used to verify the data integrity as well as several other security properties, like self-certification, name persistency, and owner authentication and identification. The approach also gives some benefits over the security model in today's host centric networks, as it minimizes the need for trust in the infrastructure, including the hosts providing the data, the channel, or the resolution service.

In NetInf the information objects are published into the network. They are registered with a Name Resolution Service (NRS). The NRS is also used to register network locators that can be used to retrieve data objects that represent the published IOs. When a receiver wants to retrieve an IO, the request for the IO is resolved by the NRS into a set of locators. These locators are then used to retrieve a copy of the data object from the "best" available source(s). NetInf is open to use any type of underlying transport networks. The locators can thus be a heterogeneous set, e.g., IPv4, IPv6, MAC, etc.

NetInf will make extensive use of caching of information objects in the network and will provide network functionality that is similar to what overlay solutions like Content Distribution Networks (CDN) and p2p distribution networks (e.g., BitTorrent) provide today.

4.7.1. Applicability to DECADE

An example of an experimental information centric network architecture that will require storage space for storage and caching of information objects on a large number of NetInf nodes in the Internet.

4.7.2. Data Access Interface

Users will publish IOs with specific IDs into the network. This is done by the client sending a register message to the NRS stating that the IO with the specific ID is available. When another user wishes to retrieve the IO they will use the given ID to make a request for the IO. The ID is then resolved by the NRS and the IO is delivered from a nearby in-network storage location.

4.7.3. Data Management Operations

Users do not have the direct ability to delete content stored in the NetInf nodes. However, there can be some type of "Time To Live" parameter associated with the information objects though this has not yet been specified.

4.7.4. Data Search Capability

Not provided.

4.7.5. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private. The basic security mechanism in NetInf is for the publisher to digitally sign the content of the information object that it publish. It is envisioned that a complete access control system can be built on top of this though this has not yet been specified.

4.7.6. Resource Control Interface

Not provided.

4.7.7. Discovery Mechanism

NetInf IDs are used for naming and accessing information objects. The IDs are resolved by the NRS into locators that are used for routing and transport of data through the transport networks. This is still an active area of research.

4.7.8. Storage Mode

Object-based. From an application perspective NetInf can be used for publishing entire files or chunks of files. NetInf is agnostic to the application perspective and treats everything as information objects.

4.8. Network Traffic Redundancy Elimination

Redundancy Elimination (RE) (e.g., [16]) is used for identifying and removing repeated content from network transfers. This technique has been proposed to improve network performance in many types of networks, such as ISP backbones and enterprise access links. One example of redundancy elimination proposal is SmartRE, proposed by Anand et al., which focuses on network-wide redundancy elimination. In packet-level redundancy elimination, forwarding elements are equipped with additional storage which can be used to cache data from

forwarded packets. Upstream routers may replace packet data with a fingerprint that tells a downstream router how to decode and reconstruct the packet based on cached data.

4.8.1. Applicability to DECADE

An example of an experimental in-network storage system that would require a large amount of associated packet processing at routers if it was ever deployed.

4.8.2. Data Access Interface

Redundancy-elimination are typically transparent to the user. Writing into the storage is done by transferring data that has not already been cached. Storage is read when users transmit data identical to previously-transmitted data.

4.8.3. Data Management Operations

Not provided.

4.8.4. Data Search Capability

Not provided.

4.8.5. Access Control Authorization

Access control method is public-restricted (to any client which is part of the RE network). Note that the content provider still retains control over which peers receive the requested data. The returned data is "compressed" as it is transferred within the network.

4.8.6. Resource Control Interface

Not provided. The content provider still retains control over the rate at which packets are sent to a peer. The packet size within the network may be reduced.

4.8.7. Discovery Mechanism

No discovery mechanism is necessary. Routers can use redundancy-elimination without the users' knowledge.

4.8.8. Storage Mode

Object-based, with "objects" being data from packets transmitted within the network.

4.9. OceanStore

OceanStore [17] is a storage platform developed at University of California, Berkeley, that provides globally-distributed storage. OceanStore implements a model where multiple storage providers can pool resources together. Thus, a major focus is on resiliency and self-organization and self-maintenance.

The protocol is resilient to some storage nodes being compromised by utilizing Byzantine agreement and erasure codes to store data at primary replicas.

4.9.1. Applicability to DECADE

An example of an experimental in-network storage system that provides a high degree of network resilience to failure scenarios.

4.9.2. Data Access Interface

Users may read and write objects

4.9.3. Data Management Operations

Objects may be replaced by newer versions, and multiple versions of an object may be maintained.

4.9.4. Data Search Capability

Not provided.

4.9.5. Access Control Authorization

Provided, but specifics for clients are unclear from the available references.

4.9.6. Resource Control Interface

Not provided.

4.9.7. Discovery Mechanism

Users require an entry-point into the system in the form of one storage node that is part of OceanStore. If a hostname is provided, the address of a storage node may be determined via DNS.

4.9.8. Storage Mode

Object-based.

4.10. Photo Sharing

There are a growing number of popular on line Photo Sharing (storing) systems. For example, the Kodak Gallery system [18] serves over 60 million users and stores billions of images [19]. Other well known examples of Photo Sharing systems include Flickr [20] and ImageShack [21]. Also there are a number of popular blogging services, such as Tumblr [22], which specialize in also sharing large numbers of photos and other multimedia content (e.g., video, text, audio, etc.) as part of their service. All these in-network storage systems utilize both free and paid subscription models.

Most Photo Sharing systems are traditional client-server architecture. However, a minority of systems also offer a P2P mode of operation. The client-server architecture is typically based on HTTP with a browser client and a web server.

4.10.1. Applicability to DECADE

Very widely used (deployed) example of in-network storage where the end user has direct visibility and extensive control of the system. Typical end user interface is through a HTTP based web browser.

4.10.2. Data Access Interface

Users can read (view) and write (store) photos.

4.10.3. Data Management Operations

Users can delete previously stored photos.

4.10.4. Data Search Capability

Users can tag photos and/or organize them using sophisticated web photo album generators. Users can then search for objects (photos) matching desired criteria.

4.10.5. Access Control Authorization

Access control method for clients is typically either private or public-unrestricted. For example, writing (storing) to a Photo blog is typically private to the owner of the account. However, all other clients can view (read) the contents of the blog (i.e., public-unrestricted). Some photo sharing websites provide private access to

read photos to allow sharing with a limited set of friends.

4.10.6. Resource Control Interface

Not provided.

4.10.7. Discovery Mechanism

Usually by manually logging on to a central web page for the service and entering the appropriate information to access the desired information. The address to which the client connects is usually determined by DNS using the hostname from the provided URL.

4.10.8. Storage Mode

File-based. Photos are usually stored as files. They can then be organized into meta-structures (e.g., albums, galleries, etc.) using sophisticated web photo album generators.

4.11. P2P Cache

Caching of P2P traffic is a useful approach to reduce P2P network traffic, because objects in P2P systems are mostly immutable and the traffic is highly repetitive. In addition, making use of P2P caches do not require changes to P2P protocols and can be deployed transparently from clients.

P2P caches operate similarly to web caches, in that they temporarily store frequently-requested content. Requests for content already stored in the cache can be served from local storage instead of requiring the data to be transmitted over expensive network links.

Two types of P2P caches exist: non-transparent P2P caches and transparent P2P caches. A non-transparent cache appears as a super peer; it explicitly peers with other P2P clients. For a transparent cache, once a P2P cache is established, the network will transparently redirect P2P traffic to the cache, which either serves the file directly or passes the request on to a remote P2P user and simultaneously caches that data. Transparency is typically implemented using deep packet inspection (DPI). DPI products identify and pass P2P packets to the P2P caching system so it can cache the traffic and accelerate it.

To enable operation with existing P2P software, P2P caches directly support P2P application protocols. A large number of P2P protocols are used by P2P software, and hence are supported by caches, leading to higher complexity. Additionally, these protocols evolve over time, and new protocols are introduced.

4.11.1. Applicability to DECADE

An example of in-network storage for P2P systems. However, unlike DECADE, the existence and operation of the storage is totally transparent to the end user.

4.11.2. Transparent P2P Caches

4.11.2.1. Data Access Interface

Data Access Interface allows P2P content to be cached (stored) and supplied (retrieved) locally such that network traffic is reduced, but it is transparent to P2P users, and P2P users implicitly use the data-access interface (in the form of their native P2P application protocol) to store or retrieve content.

4.11.2.2. Data Management Operations

Not provided.

4.11.2.3. Data Search Capability

Not provided.

4.11.2.4. Access Control Authorization

Access control method is typically public-restricted (to any client which is part of the P2P channel or swarm).

4.11.2.5. Resource Control Interface

Not provided.

4.11.2.6. Discovery Mechanism

Use of Deep Packet Inspection (DPI) means no discovery mechanism is provided to P2P users, it is transparent to P2P users. Since DPI is used to recognize P2P applications' private protocols, P2P Cache implementations must be updated as new applications are added and existing protocols evolve.

4.11.2.7. Storage Mode

Object-based. Chunks (typically, the unit of transfer amongst P2P clients) of content are stored in the cache.

4.11.3. Non-transparent P2P Caches

4.11.3.1. Data Access Interface

Data Access Interface allows P2P content to be cached (stored) and supplied (retrieved) locally such that network traffic is reduced. P2P users implicitly store and retrieve from the cache using the P2P application's native protocol.

4.11.3.2. Data Management Operations

Not provided.

4.11.3.3. Data Search Capability

Not provided.

4.11.3.4. Access Control Authorization

Access control method is typically public-restricted (to any client which is part of the P2P channel or swarm)

4.11.3.5. Resource Control Interface

Not provided.

4.11.3.6. Discovery Mechanism

A cache pretends to be normal peers to join the P2P overlay network. Other P2P users can find these cache nodes through overlay routing mechanism, just looking to them as normal neighbor nodes.

4.11.3.7. Storage Mode

Object-based. Chunks (typically, the unit of transfer amongst P2P clients) of content are stored in the cache.

4.12. Usenet

Usenet is a distributed Internet based discussion (message) system. The Usenet messages are arranged as a set of "newsgroups" that are classified hierarchically by subject. Usenet information is distributed and stored among a large conglomeration of servers that store and forward messages to one another in so called news feeds. Individual users may read messages from and post messages to a local news server typically operated by an ISP. This local server communicates with other servers and exchanges articles with them. In this fashion, the message is copied from server to server and

eventually reaches every server in the network [23].

Traditional Usenet as described above operates as a P2P network between the servers, and in a client-server architecture between the user and their local news server. The user requires a Usenet client to be installed on their computer and a Usenet server account (through their ISP). However, with the rise of web browsers the Usenet architecture is evolving to be web based. The most popular example of this is Google Groups where Google hosts all the newsgroups and client access is via a standard HTTP based web browser [24].

4.12.1. Applicability to DECADE

A historically very important and widely used (deployed) example of in-network storage in the Internet. The use of this system is rapidly declining but efforts have been made to preserve the stored content for historical purposes.

4.12.2. Data Access Interface

Users can read and post (store) messages.

4.12.3. Data Management Operations

Users sometimes have limited ability to delete messages that they previously posted.

4.12.4. Data Search Capability

Traditionally, users could manually search through the newsgroups as they are classified hierarchically by subject. In the newer web based systems there is also automatic search capability based on key word matches.

4.12.5. Access Control Authorization

Access control method is either public-unrestricted or private (to client members of that newsgroup).

4.12.6. Resource Control Interface

Not provided.

4.12.7. Discovery Mechanism

Usually by manually logging on to the Usenet account. DNS may be used to resolve hostnames to their corresponding addresses.

4.12.8. Storage Mode

File system. Messages are usually stored as files which are then organized hierarchically by subject into newsgroups.

4.13. Web Cache

Web cache [25] has been widely deployed by many ISPs to reduce bandwidth consumption and web access latency since the late 1990s. A web cache can cache the web documents (e.g., HTML pages, images) between server and client to reduce bandwidth usage, server load, and perceived lag. A web cache server is typically shared by many clients, and stores copies of documents passing through it; subsequent requests may be satisfied from the cache if certain conditions are met.

Another form of cache is a client-side cache, typically implemented in web browsers. A client side cache can keep a local copy of all pages recently displayed by browser, and when the user returns to one of these web pages, the local cached copy is reused.

A related protocol for P2P applications to use web cache is HPTP (HTTP based Peer to Peer) [26]. It proposes to share chunks of P2P files/streams using HTTP protocol with cache-control headers.

4.13.1. Applicability to DECADE

Very widely used (deployed) example of in-network storage for the key Internet application of Web browsing. The existence and operation of the storage is transparent to the end user in most cases. The content caching time is controlled by Time To Live parameters associated with the original content. The principle of web caching is to speed up web page reading by using (the same) content previously requested by a preceding user to service a new user.

4.13.2. Data Access Interface

Users explicitly read from a web cache by making requests, but they cannot explicitly write data into it. Data is implicitly stored into the web cache by requesting content that is not already cached and meets policy restrictions of the cache provider.

4.13.3. Data Management Operations

Not provided.

4.13.4. Data Search Capability

Not provided.

4.13.5. Access Control Authorization

Access control method for clients is public-unrestricted. It is important to note that if content is authenticated or encrypted (e.g., HTTPS, SSL) it will not be cached. Also if the content is flagged as private (vs public) at the HTTP level by the origin server it will not be cached.

4.13.6. Resource Control Interface

Not provided.

4.13.7. Discovery Mechanism

Web Caches can be transparently deployed between Web Server and Web Clients, employing DPI for discovery. Alternatively, web caches could be explicitly discovered by clients using techniques such as DNS or manual configuration.

4.13.8. Storage Mode

Object based. Web content is keyed within the cache by HTTP Request fields, such as Method, URI, and Headers.

4.14. Observations Regarding In-Network Storage Systems

This following observations about the surveyed In-Network storage systems are made in the context of DECADE as defined by [1].

The majority of the surveyed systems were designed for client-server architectures and do not support P2P. However, there are some important exceptions, especially for some of the newer technologies such as BranchCache and P2P Cache which do support a P2P mode.

The P2P cache systems are interesting since they do not require changes to P2P applications themselves. However, this is also a limitation in that they are required to support each application protocol.

Many of the surveyed systems were designed for caching as opposed to long term network storage. Thus during DECADE protocol design it should be carefully considered if a caching mode should be supported in addition to a long term network storage mode. There is typically a trade-off between providing a caching mode and long-term (and

usually also reliable) storage with regards to some performance metrics. Note that [1] identifies issues with classical caching from a DECADE perspective such as the fact that P2P caches typically do not allow users to explicitly control content stored in the cache.

Certain components of the surveyed systems are outside of the scope of DECADE. For example, a protocol used for searching across multiple DECADE servers is out of scope. However, applications may still be able to implement such functionality if DECADE exposes the appropriate primitives. This has the benefit of keeping the core in-network storage systems simple, while permitting diverse applications to design mechanisms that meet their own requirements.

Today, most in-network storage systems follow some variant of the authorization model of public-unrestricted, public-restricted, and private. For DECADE, we may need to evolve the authorization model to support a resource owner (e.g., end user) authorization, in addition to the network authorization.

5. Storage And Other Related Protocols

This section surveys existing storage and other related protocols, as well as comments on the usage of these protocols to satisfy DECADE's use cases. The surveyed protocols are listed alphabetically.

5.1. HTTP

HTTP [27] is a key protocol for the World Wide Web. It is a stateless client-server protocol that allows applications to be designed using the Representational State Transfer (REST) model. HTTP is often associated with downloading (reading) content from web servers to web browsers, but it also has support for uploading (writing) of content to web servers. It has been used as the underlying protocol for other protocols such as WebDAV.

HTTP is used in some of the most popular in-network storage systems surveyed previously including CDNs, Photo Sharing, and Web Cache. Usage of HTTP by a storage protocol implies that no extra SW is required in the client (i.e., web based client) as all standard Web browsers are based on HTTP.

5.1.1. Data Access Interface

Basic read and write operations are supported (using HTTP GET, PUT and POST methods).

5.1.2. Data Management Operations

Not provided.

5.1.3. Data Search Capability

Not provided

5.1.4. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private.

The majority of web pages are public-unrestricted in terms of reading but do not allow any uploading of content. In-network storage systems range from private or public-unrestricted for Photo Sharing described in Section 4.10.5 to public-unrestricted for Web Caching described in Section 4.13.5.

5.1.5. Resource Control Interface

Not provided.

5.1.6. Discovery Mechanism

Manual configuration is typically used. Clients typically address HTTP servers by providing a hostname, which is resolved to an address using DNS.

5.1.7. Storage Mode

HTTP is a protocol, it thus does not define a Storage Mode. However, a non-collection resource can typically be thought of as a "file"). These files may be organized into collections, which typically map on to the HTTP Path hierarchy, creating the illusion of a file system.

5.1.8. Comments

HTTP is based on a client-server architecture and thus is not directly applicable for the DECADE focus on P2P. Also, HTTP offers only a rudimentary toolset for storage operations compared to some of the other storage protocols.

5.2. iSCSI

Small Computer System Interface (SCSI) is a set of protocols enabling communication with storage devices such as disk drives and tapes; internet SCSI (iSCSI) [28] is a protocol enabling SCSI commands to be

sent over TCP. As in SCSI, iSCSI allows an Initiator to send commands to a Target. These commands operate on the device level as opposed to individual data objects stored on the device.

5.2.1. Data Access Interface

Read and write commands indicate which data is to be read or written by specifying the offset (using Logical Block Addressing) into the storage device. The size of data to be read or written is an additional parameter in the command.

5.2.2. Data Management Operations

Since commands operate at the device level, management operations are different than with traditional file systems. Management commands for SCSI/iSCSI including explicit device control such as starting and stopping the device and formatting the device.

5.2.3. Data Search Capability

SCSI/iSCSI does not provide the ability to search for particular data within a device. Note that such capabilities can be implemented outside of iSCSI.

5.2.4. Access Control Authorization

With respect to access to devices, the access control method is private. iSCSI uses CHAP [29] to authenticate initiators and targets when accessing storage devices. However, since SCSI/iSCSI operates at the device level, neither authentication nor authorization are provided for individual data objects. Note that such capabilities can be implemented outside of iSCSI.

5.2.5. Resource Control Interface

Not provided.

5.2.6. Discovery Mechanism

Manual configuration may be used. An alternative is the internet Storage Name Service (iSNS) [30] provides the ability to discover available storage resources.

5.2.7. Storage Mode

As a protocol, iSCSI does not explicitly have a storage mode. However, it provides block-based access to clients. SCSI/iSCSI provides an Initiator block-level access to the storage device.

5.3. NFS

The Network File System is designed to allow users to access files over a network in a manner similar to how local storage is accessed. NFS is typically used in local area network or enterprise settings, though changes made in later versions of NFS make it easier to operate over the Internet.

5.3.1. Data Access Interface

Traditional file-system operations such as read, write, and update (overwrite) are provided. Locking is provided to support concurrent access by multiple clients.

5.3.2. Data Management Operations

Traditional file-system operations such as move and delete are provided.

5.3.3. Data Search Capability

User has the ability to list contents of directories to find filenames matching desired criteria.

5.3.4. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private. For example, files and directories can be protected using read, write, and execute permissions for the files owner, group, and the public (others). Also, NFSv4.1 has a rich ACL model allowing a list of Access Control Entries (ACEs) to be configured for each file or directory. The ACEs can specify per-user read/write access to file data, file/directory attributes, creation/deletion of files in a directory, etc.

5.3.5. Resource Control Interface

While disk space quotas can be configured, it typically limits the total amount of storage allocated to a particular user. User control of bandwidth and connections used by remote peers is not provided.

5.3.6. Discovery Mechanism

Manual configuration is typically used. Clients address NFS servers by providing a hostname and a directory that should be mounted. DNS may be used lookup an address for the provided hostname.

5.3.7. Storage Mode

As a protocol, there is no defined internal storage mode. However, implementations typically use the underlying filesystem storage. Note that extensions have been defined for alternate storage modes (e.g., block-based [32] and object-based [33]).

5.3.8. Comments

The efficiency and scalability of the NFS access control method is a concern in the context of DECADE. In particular, Section 6.2.1 of [31] states that:

> Only ACEs that have a "who" that matches the requester are considered.

Thus, in the context of DECADE, to specify per-peer access control policies for an object, a client would need to explicitly configure the ACL for the object for each individual peer. A concern with this approach is scalability when a client's peers may change frequently and ACLs for many small objects need to be updated constantly during participation in a swarm.

Note that NFS v4.1's usage of RPCSEC_GSS provides support for multiple security mechanisms. Kerberos V5 is required, but others such as X.509 certificates are also supported by way of GSS-API. Note, however, that NFSv4.1's usage of such security mechanisms is limited to linking a requesting user to a particular account maintained by the NFS server.

5.4. OAuth

OAuth [34] is a protocol that enriches the traditional client-server authentication model for web resources. In particular, OAuth distinguishes the "client" from the "resource owner", thus enabling a resource owner to authorize a particular client for access (e.g., for a particular lifetime) to private resources.

We include OAuth in this survey so that its authentication model can be evaluated in the context of DECADE. OAuth itself, however, is not a network storage protocol.

5.4.1. Data Access Interface

Not provided.

5.4.2. Data Management Operations

Not provided.

5.4.3. Data Search Capability

Not provided.

5.4.4. Access Control Authorization

Not provided. While similar in spirit to the WebDAV ticketing extensions [35], OAuth instead uses the following process: (1) a client constructs a delegation request, (2) the client forwards the request to the resource owner for authorization, (3) the resource owner authorizes the request, and finally (4) a callback is made to the client indicating that its request has been authorized.

Once the process is complete, the client has a set of token credentials that grant it access to the protected resource. The token credentials may have an expiration time, and they can also be revoked by the resource owner at any time.

5.4.5. Resource Control Interface

Not provided.

5.4.6. Discovery Mechanism

Not provided.

5.4.7. Storage Mode

Not provided.

5.4.8. Comments

The ticketing mechanism requires server involvement and the discussion relating to WebDAV's proposed ticketing mechanism (see Section 5.5.8) applies here as well.

5.5. WebDAV

WebDAV [36] is a protocol designed for Web content authoring. It is developed as an extension to HTTP described in Section 5.1, meaning it can be simpler to integrate into existing software. WebDAV supports traditional operations for reading/writing from storage, as well as other constructs such as locking and collections which are important when multiple users collaborate to author or edit a set of

documents.

5.5.1. Data Access Interface

Traditional read and write operations are supported (using HTTP GET and PUT methods, respectively). Locking is provided to ease concurrent access by multiple clients.

5.5.2. Data Management Operations

WebDAV supports traditional file-system operations such as move, delete and copy. Objects are organized into collections, and these operations can also be performed on collections. WebDAV also allows objects to have user-defined properties.

5.5.3. Data Search Capability

User has the ability to list contents of collections to find objects matching desired criteria. A SEARCH extension [37] has also been specified allowing listing of objects matching client-defined criteria.

5.5.4. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private.

For example, an ACL extension [38] is provided for WebDAV. ACLs allow both user- and group-based access control policies (relating to reading, writing, properties, locking, etc) to be defined for objects and collections.

A ticketing extension [35] has also been proposed, but has not progressed beyond an Internet Draft. This extension allows a client to request the WebDAV server to create a "ticket" (e.g., for reading an object) that can be distributed to other clients. Tickets may be given expiration times, or may only allow for a fixed number of uses. The proposed extension requires the server to generate tickets and maintain state for outstanding tickets.

5.5.5. Resource Control Interface

An extension [39] allows disk space quotas to be configured for Collections. The extension also allows WebDAV clients to query current disk space usage. User control of bandwidth and connections used by remote peers is not provided.

5.5.6. Discovery Mechanism

Manual configuration is typically used. Clients address WebDAV servers by providing a hostname, which can be resolved to an address using DNS.

5.5.7. Storage Mode

Though no storage mode is explicitly defined, WebDAV can be thought of as providing file-based storage to a client. A non-collection resource can typically be thought of as a "file". Files may be organized into collections, which typically map on to the HTTP Path hierarchy.

5.5.8. Comments

The efficiency and scalability of the WebDAV access control method is a concern in the context of DECADE, for similar reasons as stated in Section 5.3.8 for NFS. The proposed WebDAV ticketing extension partially alleviates this concern, but the particular technique may need further evaluation before being applied to DECADE. In particular, since DECADE clients may continuously upload/download a large number of small-size objects, and a single DECADE server may need to scale to many concurrent DECADE clients, requiring the server to maintain ticket state and generate tickets may not be the best design choice. Server-generated tickets can also increase latency for data transport operations depending on the message flow used by DECADE.

5.6. Observations Regarding Storage and Related Protocols

This following observations about the surveyed storage and related protocols are made in the context of DECADE as defined by [1].

All of the surveyed protocols were primarily designed for client-server architectures and not for P2P. However, it is conceivable that some of the protocols could be adapted to work in a P2P architecture.

Several popular in-network storage systems today use HTTP as their key protocol even though it is not classically considered as a storage protocol. HTTP is a stateless protocol that is used to design RESTful applications. HTTP is a well supported and widely implemented protocol which can provide important insights for DECADE.

The majority of the surveyed protocols do not support low latency access for applications such as live streaming. This was one of the key general requirements for DECADE.

The majority of the surveyed protocols do not support any form of resource control interface. Resource control is required for users to manage the resources on in-network storage that can be used by other peers, e.g., the bandwidth or connections. Resource control is a key capability required for DECADE.

Nearly all surveyed protocols did however support the following capabilities which are required for DECADE: user ability to read/ write content; some form of access control; some form of error indication; and ability to traverse firewalls and NATs.

6. Conclusions

Though there have been many successful in-network storage systems, they have been designed for use cases different from those defined in DECADE. For example, many of the surveyed in-network storage systems and protocols were designed for client-server architectures and not P2P. No surveyed system or protocol has the functionality and features to fully meet the set of requirements defined for DECADE. DECADE aims to provide a standard protocol for P2P applications and content provider to access and control in-network storage, resulting in increased network efficiency while retaining control over content shared with peers. Additionally, defining a standard protocol can reduce complexity of in-network storage since multiple P2P application protocols no longer need to be implemented by in-network storage systems.

7. Security Considerations

This draft is a survey of existing in-network storage systems, and does not introduce any security considerations beyond those of the surveyed systems.

For more information on security considerations of DECADE, see [1].

8. IANA Considerations

This document does not have any IANA Considerations.

9. Contributors

The editors would like to thank the following people for contributing to the development of this document:

- ZhiHui Lu

- Borje Ohlman

- Pang Tao

- Lucy Yong

- Juan Carlos Zuniga

10. Acknowledgments

The editors would like to thank the following people for providing valuable comments to various versions of this document: David Bryan, Tao Mao, Haibin Song, Ove Strandberg, Yu-Shun Wang, Richard Woundy, Yunfei Zhang, and Ning Zong.

11. Informative References

[1] Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", draft-ietf-decade-problem-statement-02 (work in progress), January 2011.

[2] Storage Search, "Flash Memory vs. Hard Disk Drives - Which Will Win?", http://www.storagesearch.com/semico-art1.html.

[3] Matt's Computer Trends, "Flash and Disk Trends", http://www.mattscomputertrends.com/flashdiskcomparo.html.

[4] Yingjie, G., Bryan, D., Yang, Y., and R. Alimi, "DECADE Requirements", draft-ietf-decade-reqs-00 (work in progress), October 2010.

[5] Amazon, "Amazon Simple Storage Service (Amazon S3)", http://aws.amazon.com/s3/.

[6] Microsoft Corporation, "Windows Azure Blob - Programming Blob Storage".

[7] Google, "Google Storage for Developers", http://code.google.com/apis/storage.

[8] Microsoft Corporation, "BranchCache", http://technet.microsoft.com/en-us/network/dd425028.aspx.

[9] S. Paul, R. Yates, D. Raychaudhuri, J. Kurose., "The Cache-and-Forward Network Architecture for Efficient Mobile Content Delivery Services in the Future Internet", In Innovations in NGN: Future Network and Services, 2008.

[10] Pathan, A.K., Buyya, R., "A Taxonomy and Survey of Content Delivery Networks", In Grid Computing and Distributed Systems Laboratory in University of Melbourne, Technology Report, Feb. 2007.

[11] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

[12] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

[13] Named Data Networking, "Named Data Networking Home Page", http://www.named-data.net/.

[14] Named Data Networking, "Named Data Networking Project Proposal", http://www.named-data.net/ndn-proj.pdf.

[15] Network of Information., "Network of Information Overview", http://www.netinf.org/home/overview/.

[16] A. Anand, V. Sekar, A. Akella., "SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination", In SIGCOMM 2009.

[17] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz., "Pond: the OceanStore Prototype", In FAST 2003.

[18] Kodak, "Kodak Gallery Home Page", http://www.kodakgallery.com/gallery/welcome.jsp.

[19] Wikipedia, "Kodak Gallery", http://en.wikipedia.org/wiki/Kodak_Gallery.

[20] Flickr, "Flickr Home Page", http://www.flickr.com.

[21] ImageShack, "ImageShack Home Page", http://imageshack.us.

[22] Tumblr, "Tumblr Home Page", http://www.tumblr.com.

[23] Wikipedia, "Usenet", http://en.wikipedia.org/wiki/Usenet.

[24] Google, "Google Groups", http://groups.google.com.

[25] Geoff Huston, Telstra., "Web Caching", In The Internet Protocol Journal Volume 2, No. 3.

[26] G. Shen, Y. Wang, Y. Xiong, B.Y. Zhao, Z.-L. Zhang, "HPTP: Relieving the tension between isps and p2p", In 6th International workshop on Peer-To-Peer Systems (IPTPS2007).

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

[28] Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M., and E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)", RFC 3720, April 2004.

[29] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[30] Tseng, J., Gibbons, K., Travostino, F., Du Laney, C., and J. Souza, "Internet Storage Name Service (iSNS)", RFC 4171, September 2005.

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

[32] Black, D., Fridella, S., and J. Glasgow, "Parallel NFS (pNFS) Block/Volume Layout", RFC 5663, January 2010.

[33] Halevy, B., Welch, B., and J. Zelenka, "Object-Based Parallel NFS (pNFS) Operations", RFC 5664, January 2010.

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

[35] Ito, K., "Ticket-Based Access Control Extension to WebDAV", draft-ito-dav-ticket-00 (work in progress), October 2001.

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

[37] Reschke, J., Reddy, S., Davis, J., and A. Babich, "Web Distributed Authoring and Versioning (WebDAV) SEARCH", RFC 5323, November 2008.

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

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

Authors' Addresses

Richard Alimi (editor)
Google

Email: ralimi@google.com

Akbar Rahman (editor)
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

Yang Richard Yang (editor)
Yale University

Email: yry@cs.yale.edu

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

R. Alimi, Ed.
Google
A. Rahman, Ed.
InterDigital Communications, LLC
Y. Yang, Ed.
Yale University

August 31, 2011

A Survey of In-network Storage Systems
draft-ietf-decade-survey-06

Abstract

This document surveys deployed and experimental in-network storage systems and describes their applicability for the DECADE (DECoupled Application Data Enroute) architecture.

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 March 3, 2012.

Table of Contents

1. Introduction

DECADE (DECoupled Application Data Enroute) is an architecture that provides applications with access to provider based in-network storage for content distribution (hereafter referred to as only "in-network storage" in this document). With access to in-network storage, content distribution applications can be designed to place less load on network infrastructure. 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. See [1] for further discussion.

A major motivation for DECADE is the substantial increase of capacity and reduction in cost offered by storage systems. For example, over the last two decades, there has been at least a 30-fold increase in the amount of storage that you can get for a given price (for flash memory and hard disk drives) [2], [3].

High-capacity and low-cost in-network storage devices introduce substantial opportunities. One example of in-network storage is content caches supporting Web and Peer-to-Peer (P2P) content. Different from existing content caches whose control fully reside at the owners of the caching devices, DECADE also allows applications to control access to their allocated in-network storage, as well as the resources consumed while accessing that storage (bandwidth, connections, storage space). While designed in the context of P2P applications, it may be useful to other applications as well. This document provides details on deployed and experimental in-network storage solutions, and evaluates their suitability for DECADE.

We note that the survey presented in this document is only representative of the research in this area. Rather than trying to enumerate an exhaustive list, we have chosen some typical techniques that lead to derivative works.

2. Survey Overview

2.1. Terminology and Concepts

This document uses terms defined in [1].

2.2. Historical Context

In-network storage has been used previously in numerous scenarios to reduce network traffic and enable more efficient content distribution. This section presents a brief history of content distribution techniques and illustrates how DECADE relates to past

approaches. Systems have been developed with particular use cases in mind. Thus, this survey is not meant to point out shortcomings of existing solutions, but rather to indicate where certain capabilities required in DECADE [4] are not provided by existing systems.

In the early stage of Internet development, most Web content was stored at a central server and clients requested Web content from the central server. In this architecture, the central server was required to provide a large amount of bandwidth. Web browsing is still a primary activity on today's Internet. As more and more users access Web content, a central server can become overloaded. The use of web caches is one technique to reduce load on a central server. Web caches store frequently-requested content, and provide bandwidth for serving the content to clients.

The ongoing growth of broadband technology in the worldwide market has been driven by the hunger of customers for new multimedia services as well as Web content. In particular, the use of audio and video streaming formats has become common for delivery of rich information to the public, both residential and business.

To overcome this challenge of massive multimedia consumption, just installing more Web cache will not be enough. Moving content closer to the consumer results in greater network efficiency, improved QoS, and lower latency, while facilitating personalization of content through broadband content applications. In these edge technologies, Content Delivery Networks (CDNs) is a representative technique. CDNs are based on a large-scale distributed network of servers located closer to the edges of the Internet for efficient delivery of digital content including various forms of multimedia content.

Although CDN is an effective means of information access and delivery, there are two barriers to making CDN a more common service: cost and replication integrity. Deploying a CDN with its associated infrastructure is expensive. A CDN also requires administrative control over nodes with large storage capacity at geographically dispersed locations with adequate connectivity. CDN can be scalable, but due to this administrative and cost overhead, not rapidly deployable for the common user.

The emergence and maturation of P2P has allowed improvements to many network applications. P2P allows the use of client resources, such as CPU, memory, storage, and bandwidth, for serving content. This can reduce the amount of resources required by a content provider. Multimedia content delivery using various P2P or peer-assisted frameworks has been shown to greatly reduce the dependence on CDN and central content servers. However, the popularity of P2P applications has resulted in increased traffic on ISP networks. P2P caches (both

transparent and non-transparent) have been introduced as a way to reduce the burden. Though they can be effective in reducing traffic in certain areas of ISP networks, P2P caches have their shortcomings. In particular, they are application-dependent and thus difficult to keep up-to-date with new and evolving P2P application protocols. Second, applications may benefit from explicit control of in-network storage, which P2P caches do not provide. See [1] for further discussion.

DECADE aims to provide a standard protocol allowing P2P applications (including Content Providers) to make use of in-network storage to reduce the traffic burden on ISP networks, while enabling P2P applications to control access to content they have placed in in-network storage.

3. In-network Storage System Components

Before surveying individual technologies, we describe the basic components of in-network storage. For consistency and for ease of comparison, we use the same model to evaluate each storage technology in this document.

Note that the network protocol(s) used by a given storage system are also an important part of the design. We omit details of particular protocol choices in this document.

3.1. Data Access Interface

A set of operations available to a client user for accessing data in the in-network storage. Solutions typically allow both read and write, though the mechanisms for doing so can differ drastically.

3.2. Data Management Operations

Storage systems may provide users the ability to manage stored content. For example, operations such as delete and move may be provided to users. In this survey, we focus on data management operations that are provided to client users and omit those provided to system administrators.

3.3. Data Search Capability

Some storage systems may provide the capability to search or enumerate content that has been stored. In this survey, we focus on search capabilities that are provided to client users and omit those provided to system administrators. An example of a client search would be to find out the list of items stored by the given user over

a given period of time.

3.4. Access Control Authorization

Storage systems typically allow a client user, content owner or some other entity to define the access policies for the in-network storage. The in-network storage system then checks the authorization of a user before it stores or retrieves content. We define three types of access control authorization: public-unrestricted, public-restricted, and private.

Public-unrestricted refers to content on an in-network storage system that is widely available to all clients (i.e., without restrictions). An example is accessing Wikipedia on the Web, or anonymous access to FTP sites.

Public-restricted refers to content on an in-network storage system that is available to a restricted (though still potentially large) set of clients, but which do not require any confidential credentials from the client. An example is some content (e.g., a TV show episode) on the Internet that can only be viewable in selected countries or networks (i.e., white/black lists or black-out areas).

Private refers to content on an in-network storage system that is only made available to one or more clients presenting the required confidential credentials (e.g., password or key). This content is not available to anyone without the proper confidential access credentials.

Note that a combination of access control types may be applicable for a given scenario. For example, the retrieval (read) of content from an in-network storage system may be public-unrestricted, but the storage (write) to the same system may be private.

3.5. Resource Control Interface

This is the interface through which users manage the resources on in-network storage that can be used by other peers, e.g., the bandwidth or connections. The storage system may also allow users to indicate a time for which resources are granted.

3.6. Discovery Mechanism

Users use the discovery mechanism to find location of in-network storage, find access interface or resource control interface or other interfaces of in-network storage.

3.7. Storage Mode

Storage systems may use the following modes of storage: file system, object-based, or block-based.

A file system typically organizes files into a hierarchical tree structure. Each level of the hierarchy normally contains one or more directories each with one or more files. A file system may also be flat or use some other organizing principle.

We define an object-based storage mode as one which stores discrete chunks of data (e.g., IP datagrams or another type of aggregation useful to an application) without a pre-defined hierarchy or meta structure.

We define a block-based storage mode as one which stores a raw sequence of bytes, with a client being able to read and/or write data at offsets within that sequence. Data is typically accessed in blocks for efficiency. An common example for this storage mode is raw access to a hard disk.

In this survey, we define Storage Mode to refer to how data is structured within the system, which may not be the same as how it is accessed by a client. For example, a caching system may cache objects with hierarchical names, but may internally use an object-based Storage Mode.

4. In-Network Storage Systems

This section surveys in-network storage systems using the methodology defined above. The survey includes some systems that are widely deployed today, some systems that are just being deployed, and some experimental/futuristic systems. The survey covers both traditional client-server architectures and P2P architectures. The surveyed systems are listed in alphabetical order. Also, for each system, a brief explanation is given of the relevance to DECADE.

4.1. Amazon S3

Amazon S3 (Simple Storage Service) [5] provides an online storage service using web (HTTP) interfaces. Users create buckets, and each bucket can contain stored objects. Users are provided an interface through which they can manage their buckets. Amazon S3 is a popular backend storage for other services. Other related storage services is the Blob Service provided by Windows Azure [6], Google Storage for Developers [7], and Dropbox [8].

4.1.1. Applicability to DECADE

Very widely used (deployed) example of in-network storage. Amazon leases the storage to third party companies for disparate services. In particular, Amazon S3 has a rich model for authorization (using signed queries) to integrate with a wide variety of use cases. A focus for Amazon S3 is scalability. Particular simplifications that were made are the absence of a general, hierarchical namespace and the inability to update the contents of existing data.

4.1.2. Data Access Interface

Users can read, and write objects.

4.1.3. Data Management Operations

Users can delete previously-stored objects.

4.1.4. Data Search Capability

Users can list contents of buckets to find objects matching desired criteria.

4.1.5. Access Control Authorization

All methods of access control are supported for clients: public-unrestricted, public-restricted and private.

For example, access to stored objects can be restricted by owner, a list of other Amazon Web Service users, all Amazon Web Service Users, or open to all users (anonymous access). Another option is for the owner to generate and sign a query (e.g., a query to read an object) that can be used by any user until an owner-defined expiration time.

4.1.6. Resource Control Interface

Not provided.

4.1.7. Discovery Mechanism

Users are provided a well-known DNS name (either a default provided by Amazon, or one customized by a particular user). Users accessing S3 storage use DNS to discover an IP address where S3 requests can be sent.

4.1.8. Storage Mode

Object-based, with the extension that objects can be organized into user-defined buckets.

4.2. BranchCache

BranchCache [9] is a feature integrated into Windows (Windows 7 and Windows Server 2008R2) that aims to optimize enterprise branch office file access over the WAN links. The main goals are to reduce WAN link utilization and improve application responsiveness by caching and sharing content within a branch while still maintaining end-to-end security. BranchCache allows files retrieved from the web servers and file servers located in headquarters or datacenters to be cached in remote branch offices, and shared among users in the same branch accessing the same content. BranchCache operates transparently by instrumenting the HTTP and SMB components of the networking stack. It provides two modes of operation: Distributed Cache and Hosted Cache.

In both modes, a client always contacts a BranchCache-enabled content server first to get the content identifiers for local search. If the content is cached locally, the client then retrieves the content within the branch. Otherwise, the client will go back to the original content server to request the content. The two modes differ in how the content is shared.

In the Hosted Cache mode, a locally provisioned server acts as a cache for files retrieved from the servers. After getting the content identifiers, the client first consults the cache for the desired file. If it is not present in the cache, the client retrieves it from the content server and sends it to the cache for storage.

In the Distributed Cache mode, a client first queries other clients in the same network using the Web Services Discovery multicast protocol. As in the Hosted Cache mode, the client retrieves the file from the content server if it is not available locally. After retrieving the file (either from another client or the content server), the client stores the file locally.

The original content server always authorizes requests from clients. Cached content is encrypted such that only clients can decrypt the data using keys derived from metadata returned by the content server. In addition to instrumenting the networking stack at clients, content servers must also support BranchCache.

4.2.1. Applicability to DECADE

BranchCache is an example of an in-network storage system primarily targeted at enterprise networks. It supports both a P2P like mode (Distributed Cache) as well as a client-server mode (Hosted Cache). Integration into the Microsoft OS will ensure wide distribution of this in-network storage technology.

4.2.2. Data Access Interface

Clients transparently retrieve (read) data from a cache (other clients or a Hosted Cache) since it operates by instrumenting the networking stack. In Hosted Cache mode, clients write data to the Hosted Cache once it is retrieved from the content server.

4.2.3. Data Management Operations

Not provided.

4.2.4. Data Search Capability

Not provided.

4.2.5. Access Control Authorization

Access control method for clients is private. For example, transferred content is encrypted, and can only be decrypted by keys derived from data received from the original content server. Though data may be transferred to unauthorized clients, end-to-end security is maintained by only allowing authorized clients to decrypt the data.

4.2.6. Resource Control Interface

The storage capacity of caches on the clients and Hosted Caches are configurable by system administrators. The Hosted Cache further allows configuration of the maximum number of simultaneous client accesses. In the Distributed Caching mode, exponential back-off and throttling mechanisms are utilized to prevent reply storms of popular content requests. The client will also spread data block access among multiple serving clients that have the content (complete or partial) to improve latency and provide some load balancing.

4.2.7. Discovery Mechanism

The Distributed Cache mode uses multicast for discovery of other clients and content within a local network. Currently, the Hosted Cache mode uses policy provisioning or manual configuration of the

server used as the Hosted Cache. In this mode, the address of the server may be found via DNS.

4.2.8. Storage Mode

Object-based.

4.3. Cache-and-Forward Architecture

Cache-and-Forward (CNF) [10] is an architecture for content delivery services for the future Internet. In this architecture, storage can be exploited at nodes within the network, either directly at routers or deployed nearby the routers. CNF is based on the concept of store-and-forward routers with large storage, providing for opportunistic delivery to occasionally disconnected mobile users and for in-network caching of content. The proposed CNF protocol uses reliable hop-by-hop transfer of large data files between CNF routers in place of an end-to-end transport protocol like TCP.

4.3.1. Applicability to DECADE

An example of an experimental in-network storage system that would require storage space on (or near) a large number of routers in the Internet if it was deployed. As the name of the system implies, it would provide short term caching and not long term network storage.

4.3.2. Data Access Interface

Users implicitly store content at Cache-and-forward routers by requesting files. End hosts read content from in-network storage by submitting queries for content.

4.3.3. Data Management Operations

Not provided.

4.3.4. Data Search Capability

Not provided.

4.3.5. Access Control Authorization

Access control method is public-restricted (to any client which is part of the cache-and-forward network).

4.3.6. Resource Control Interface

Not provided.

4.3.7. Discovery Mechanism

A query including a location-independent content ID is sent to the network and routed to a Cache-and-forward router, which handles retrieval of the data and forwarding to the end host.

4.3.8. Storage Mode

Object-based (with objects representing individual files). The architecture proposes to cache large files in storage within the network, though objects could be made to represent smaller chunks of larger files.

4.4. Cloud Data Management Interface

The Cloud Data Management Interface (CDMI) is a specification to access and manage cloud storage. CDMI is specified by the Storage Networking Industry Association (SNIA).

CDMI is a functional interface that applications can use to create, retrieve, update and delete data elements from the cloud. As part of this interface the client will be able to discover the capabilities of the cloud storage offering and use this interface to manage containers and the data that is placed in them. In addition, metadata can be set on containers and their contained data elements through this interface [11].

CDMI follows a traditional client server model, and operates over an HTTP interface using the Representational State Transfer (REST) model. Similar to Amazon S3 buckets (see Section 4.1), users may create containers into which data objects may be stored. Even though data objects may be accessed via a user-defined name within a container, it is also possible to access data objects by a storage-defined Object ID which is provided in the response upon creation of a Data Object.

4.4.1. Applicability to DECADE

CDMI is an important initiative to standardize storage interfaces for cloud services which are rapidly becoming an important storage service. In particular, it specifies a set of operations for creating, reading, writing, and managing data objects at a remote server (or set of servers) via the HTTP protocol.

4.4.2. Data Access Interface

Users can read and write data objects, and also update data in existing data objects. CDMI data objects are sent on the wire embeded as a field in a JavaScript Object Notation (JSON) object. The protocol also defines interfaces in which the contents of data objects can be written via simple HTTP GET/PUT operations.

4.4.3. Data Management Operations

Users can delete already-existing data objects. The create operation also supports modes in which the created object is copied or moved from an existing data object.

Data System Metadata also allows users to configure policies regarding time-to-live after which a data object is automatically deleted, as well as the redundancy with which a data object is stored.

4.4.4. Data Search Capability

Users may list the contents of containers to locate data objects matching any desired criteria.

4.4.5. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private.

In particular CDMI allows access to data objects to be protected by ACLs which can allow or restrict access based on user, group, administrative status, or whether a user is authenticated or anonymous.

4.4.6. Resource Control Interface

CDMI supports attributes 'cdmi_max_latency' and 'cdmi_max_throughput' (set at either the level of containers, or a specific data object) which control the level of service offered to any users accessing a particular data object.

4.4.7. Discovery Mechanism

Users are provided a well-known DNS name. The DNS name is resolved to determine the IP address to which requests may be sent.

4.4.8. Storage Mode

Object-based, with the extension that objects can be organized into user-defined containers.

4.5. Content Delivery Network

A CDN provides services that improve performance by minimizing the amount of data transmitted through the network, improving accessibility and maintaining correctness through content replication. CDNs offer fast and reliable applications and services by distributing content to cache or edge servers located close to users. See [12] for an additional taxonomy and survey.

A CDN has some combination of content-delivery, request-routing, distribution and accounting infrastructure. The content-delivery infrastructure consists of a set of edge servers (also called surrogates) that deliver copies of content to end-users. The request-routing infrastructure is responsible for directing client requests to appropriate edge servers. It also interacts with the distribution infrastructure to keep an up-to-date view of the content stored in the CDN caches. The distribution infrastructure moves content from the origin server to the CDN edge servers and ensures consistency of content in the caches. The accounting infrastructure maintains logs of client accesses and records the usage of the CDN servers. This information is used for traffic reporting and usage-based billing.

In practice, a CDN typically hosts static content including images, video, media clips, advertisements, and other embedded objects for Web viewing. A focus for CDNs is the ability to publish and deliver content to end-users in a reliable and timely manner. A CDN focuses on building its network infrastructure to provide the following services and functionalities: storage and management of content; distribution of content among surrogates; cache management; delivery of static, dynamic and streaming content; backup and disaster recovery solutions; and monitoring, performance measurement and reporting.

Examples of existing CDNs are Akamai, Limelight, and CloudFront.

The following description uses the term "content provider" to refer to the entity purchasing a CDN service, and the term "client" to refer to the subscriber requesting content via the CDN from the content provider.

4.5.1. Applicability to DECADE

Very widely used (deployed) example of in-network storage for multimedia content. The existence and operation of the storage is totally transparent to the end user. A CDN typically require a strong business relationship between the content providers and content distributors and often the business relationship extends to the ISPs.

4.5.2. Data Access Interface

A CDN is typically a closed system, and generally provides only read (retrieve) access interface to clients. A CDN typically does not provide write (store) access interface to clients. The content provider can access network edge servers and store content on them, or edge servers can retrieve content from content providers. Client nodes can just retrieve content from edge servers.

4.5.3. Data Management Operations

A content provider can manage the data distributed in different cache nodes, such as moving popular data objects from one cache node to another cache node, or deleting rarely-accessed data objects in cache nodes. Client user nodes, however, have no right to perform these operations.

4.5.4. Data Search Capability

A content provider can search or enumerate the data each cache node stores. Client user nodes cannot perform search operations.

4.5.5. Access Control Authorization

All methods of access control (for reading) are supported for clients: public-unrestricted, public-restricted and private. Some CDN edge servers will allow usage of HTTP basic authentication with the origin server, restrictions by IP address, or they can use a token-based technique to allow the origin server to apply its own authorization criteria.

As mentioned previously, clients typically cannot write to the CDN. Writing is typically a private operation for the content providers.

4.5.6. Resource Control Interface

Not provided.

4.5.7. Discovery Mechanism

Content providers can directly find internal CDN cache nodes to store content, since they typically have an explicit business relationship. Clients can locate CDN nodes through DNS or other redirection mechanisms.

4.5.8. Storage Mode

Though addressing objects uses URLs which typically refer to objects in a hierarchical fashion, the storage mode is typically object-based.

4.6. Delay-Tolerant Network

The Delay-Tolerant Network (DTN) [13] is an evolution of an architecture originally designed for the Interplanetary Internet. The Interplanetary Internet is a communication system envisioned to provide Internet-like services across interplanetary distances in support of deep space exploration. The DTN architecture can be utilized in various operational environments characterized by severe communication disruptions, disconnections and high-delays (e.g., a month long loss of connectivity between two planetary networks because of high solar radiation due to sun spots). The DTN architecture is thus suitable for environments including deep space networks, sensor-based networks, certain satellite networks and underwater acoustic networks.

A key aspect of the DTN is a store and forward overlay layer called the "Bundle Protocol" or "Bundle Layer" that exists between the transport and application layers [14]. The Bundle Layer forms a logical overlay that employs persistent storage to help combat long term network interruptions by providing a store and forwarding service. While traditional IP networks are also based on store and forward principles, the amount of time of a packet being kept in "storage" at a traditional IP router is typically in the order of milli-seconds (or less). In contrast, the DTN architecture assumes that most Bundle Layer nodes will use some form of persistent storage (e.g., hard disk, flash memory, etc.) for DTN packets because of the nature of the DTN environment.

4.6.1. Applicability to DECADE

An example of an experimental in-network storage system that would require fundamental changes to the Internet protocols.

4.6.2. Data Access Interface

Users implicitly cause content to be stored (until successfully forwarded) at Bundle Layer nodes by initiating/terminating any transaction that traverses the DTN.

4.6.3. Data Management Operations

Users can implicitly cause deletion of content stored at Bundle Layer nodes via a "Time To Live" type parameter that the user can control (for transactions originating from the user).

4.6.4. Data Search Capability

Not provided.

4.6.5. Access Control Authorization

Access control method is public-restricted (to any client which is part of the DTN) or private.

4.6.6. Resource Control Interface

Not provided.

4.6.7. Discovery Mechanism

A Uniform Resource Identifier (URI) approach is used as the basis of the addressing scheme for DTN transactions (and subsequent store and forward routing through the DTN network).

4.6.8. Storage Mode

Object-based. DTN applications send data to the Bundle Layer which then breaks the data into segments. These segments are then routed through the DTN network, and stored in Bundle Layer nodes as required (before being forwarded).

4.7. Named Data Networking

Named Data Networking (NDN) [15] is a research initiative which proposes to move to a new model of addressing and routing for the Internet. NDN uses "named data" based routing and forwarding, to replace the current IP address based model. NDN also uses name-based data caching in the routers.

Each NDN Data packet will be assigned a content name and will be cryptographically signed. Data delivery is driven by the requesting

end. Routers disseminate name-based prefix announcements by using routing protocols like Intermediate System to Intermediate System (IS-IS) or Border Gateway Protocol (BGP). The requester will send out an "Interest" packet which identifies the name of the data that it wants. Routers that receive this Interest packet will remember the interface it came from and will then forward it on a named-based routing protocol. Once an Interest packet reaches a node that has the desired data, a named Data packet is sent back, which carries both the name and content of the data, along with a digital signature of the producer. This named Data packet is then forwarded back to the original requester on the reverse path of the Interest packet [16].

A key aspect of NDN is that routers have the capability to cache the named data. If a request for the same data (i.e., same name) comes to the router, then the NDN router will forward the named data stored locally to fulfill the request. The proponents of NDN believe that the network can be designed naturally matching data delivery characteristics instead of communication between endpoints because data delivery has become the primary use of the network.

4.7.1. Applicability to DECADE

An example of an experimental in-network storage system that would require storage space on a large number of routers in the Internet. Named Data packets would be kept in storage in the NDN routers and provided to new requesters of the same data.

4.7.2. Data Access Interface

Users implicitly store content at NDN routers by requesting content (named Data packets) from the network. Subsequent requests by different users for the same content will cause the named Data packets to be read from the NDN routers in-network storage.

4.7.3. Data Management Operations

Users do not have the direct ability to delete content stored in the NDN routers. However, there will be some type of "Time To Live" parameter associated with the named Data packets though this has not yet been specified.

4.7.4. Data Search Capability

Not provided.

4.7.5. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private.

The basic security mechanism in NDN is for the sender to digitally sign the content (named Data packet) that it sends. It is envisioned that a complete access control system can be built on top of this though this has not yet been specified.

4.7.6. Resource Control Interface

Not provided.

4.7.7. Discovery Mechanism

Names are used as the basis of the addressing and discovery scheme for NDN (and subsequent store and forward routing through the NDN network). NDN names are assumed to be hierarchical and to be able to be deterministically constructed. This is still an active area of research.

4.7.8. Storage Mode

Object-based. NDN sends named Data packets through the network. These Data packets are routed through the NDN network, and stored in NDN routers.

4.8. Network of Information

Similar to NDN (see Section 4.7), Network of Information (NetInf) [17] is another information centric approach in which the named data objects are the basic component of the networking architecture. NetInf is thus moving away from today's host centric networking architecture where the nodes in the network are the primary objects. In today's network the information objects are named relative to the hosts they are stored on (e.g., http://www.example.com/information-object.txt).

The NetInf naming and security framework builds the foundation for an information centric security model that integrates security deeply into the architecture. In this model, trust is based on the information itself. Information Objects (IOs) are given a unique name with cryptographic properties. Together with additional metadata, the name can be used to verify the data integrity as well as several other security properties, like self-certification, name persistency, and owner authentication and identification. The approach also gives some benefits over the security model in today's

host centric networks, as it minimizes the need for trust in the infrastructure, including the hosts providing the data, the channel, or the resolution service.

In NetInf the information objects are published into the network. They are registered with a Name Resolution Service (NRS). The NRS is also used to register network locators that can be used to retrieve data objects that represent the published IOs. When a receiver wants to retrieve an IO, the request for the IO is resolved by the NRS into a set of locators. These locators are then used to retrieve a copy of the data object from the "best" available source(s). NetInf is open to use any type of underlying transport networks. The locators can thus be a heterogeneous set, e.g., IPv4, IPv6, MAC, etc.

NetInf will make extensive use of caching of information objects in the network and will provide network functionality that is similar to what overlay solutions like CDNs and P2P distribution networks (e.g., BitTorrent) provide today.

4.8.1. Applicability to DECADE

An example of an experimental information centric network architecture that will require storage space for storage and caching of information objects on a large number of NetInf nodes in the Internet.

4.8.2. Data Access Interface

Users will publish IOs with specific IDs into the network. This is done by the client sending a register message to the NRS stating that the IO with the specific ID is available. When another user wishes to retrieve the IO they will use the given ID to make a request for the IO. The ID is then resolved by the NRS and the IO is delivered from a nearby in-network storage location.

4.8.3. Data Management Operations

Users do not have the direct ability to delete content stored in the NetInf nodes. However, there can be some type of "Time To Live" parameter associated with the information objects though this has not yet been specified.

4.8.4. Data Search Capability

Not provided.

4.8.5. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private. The basic security mechanism in NetInf is for the publisher to digitally sign the content of the information object that it publish. It is envisioned that a complete access control system can be built on top of this though this has not yet been specified.

4.8.6. Resource Control Interface

Not provided.

4.8.7. Discovery Mechanism

NetInf IDs are used for naming and accessing information objects. The IDs are resolved by the NRS into locators that are used for routing and transport of data through the transport networks. This is still an active area of research.

4.8.8. Storage Mode

Object-based. From an application perspective NetInf can be used for publishing entire files or chunks of files. NetInf is agnostic to the application perspective and treats everything as information objects.

4.9. Network Traffic Redundancy Elimination

Redundancy Elimination (RE) (e.g., [18]) is used for identifying and removing repeated content from network transfers. This technique has been proposed to improve network performance in many types of networks, such as ISP backbones and enterprise access links. One example of redundancy elimination proposal is SmartRE, proposed by Anand et al., which focuses on network-wide redundancy elimination. In packet-level redundancy elimination, forwarding elements are equipped with additional storage which can be used to cache data from forwarded packets. Upstream routers may replace packet data with a fingerprint that tells a downstream router how to decode and reconstruct the packet based on cached data.

4.9.1. Applicability to DECADE

An example of an experimental in-network storage system that would require a large amount of associated packet processing at routers if it was ever deployed.

4.9.2. Data Access Interface

Redundancy-elimination are typically transparent to the user. Writing into the storage is done by transferring data that has not already been cached. Storage is read when users transmit data identical to previously-transmitted data.

4.9.3. Data Management Operations

Not provided.

4.9.4. Data Search Capability

Not provided.

4.9.5. Access Control Authorization

Access control method is public-restricted (to any client which is part of the RE network). Note that the content provider still retains control over which peers receive the requested data. The returned data is "compressed" as it is transferred within the network.

4.9.6. Resource Control Interface

Not provided. The content provider still retains control over the rate at which packets are sent to a peer. The packet size within the network may be reduced.

4.9.7. Discovery Mechanism

No discovery mechanism is necessary. Routers can use redundancy-elimination without the users' knowledge.

4.9.8. Storage Mode

Object-based, with "objects" being data from packets transmitted within the network.

4.10. OceanStore

OceanStore [19] is a storage platform developed at University of California, Berkeley, that provides globally-distributed storage. OceanStore implements a model where multiple storage providers can pool resources together. Thus, a major focus is on resiliency and self-organization and self-maintenance.

The protocol is resilient to some storage nodes being compromised by

utilizing Byzantine agreement and erasure codes to store data at primary replicas.

4.10.1. Applicability to DECADE

An example of an experimental in-network storage system that provides a high degree of network resilience to failure scenarios.

4.10.2. Data Access Interface

Users may read and write objects

4.10.3. Data Management Operations

Objects may be replaced by newer versions, and multiple versions of an object may be maintained.

4.10.4. Data Search Capability

Not provided.

4.10.5. Access Control Authorization

Provided, but specifics for clients are unclear from the available references.

4.10.6. Resource Control Interface

Not provided.

4.10.7. Discovery Mechanism

Users require an entry-point into the system in the form of one storage node that is part of OceanStore. If a hostname is provided, the address of a storage node may be determined via DNS.

4.10.8. Storage Mode

Object-based.

4.11. Photo Sharing

There are a growing number of popular on line Photo Sharing (storing) systems. For example, the Kodak Gallery system [20] serves over 60 million users and stores billions of images [21]. Other well known examples of Photo Sharing systems include Flickr [22] and ImageShack [23]. Also there are a number of popular blogging services, such as Tumblr [24], which specialize in also sharing large numbers of photos

and other multimedia content (e.g., video, text, audio, etc.) as part of their service. All these in-network storage systems utilize both free and paid subscription models.

Most Photo Sharing systems are traditional client-server architecture. However, a minority of systems also offer a P2P mode of operation. The client-server architecture is typically based on HTTP with a browser client and a web server.

4.11.1. Applicability to DECADE

Very widely used (deployed) example of in-network storage where the end user has direct visibility and extensive control of the system. Typical end user interface is through a HTTP based web browser.

4.11.2. Data Access Interface

Users can read (view) and write (store) photos.

4.11.3. Data Management Operations

Users can delete previously stored photos.

4.11.4. Data Search Capability

Users can tag photos and/or organize them using sophisticated web photo album generators. Users can then search for objects (photos) matching desired criteria.

4.11.5. Access Control Authorization

Access control method for clients is typically either private or public-unrestricted. For example, writing (storing) to a Photo blog is typically private to the owner of the account. However, all other clients can view (read) the contents of the blog (i.e., public-unrestricted). Some photo sharing websites provide private access to read photos to allow sharing with a limited set of friends.

4.11.6. Resource Control Interface

Not provided.

4.11.7. Discovery Mechanism

Usually by manually logging on to a central web page for the service and entering the appropriate information to access the desired information. The address to which the client connects is usually determined by DNS using the hostname from the provided URL.

4.11.8. Storage Mode

File-based. Photos are usually stored as files. They can then be organized into meta-structures (e.g., albums, galleries, etc.) using sophisticated web photo album generators.

4.12. P2P Cache

Caching of P2P traffic is a useful approach to reduce P2P network traffic, because objects in P2P systems are mostly immutable and the traffic is highly repetitive. In addition, making use of P2P caches does not require changes to P2P protocols and can be deployed transparently from clients.

P2P caches operate similarly to web caches (Section 4.14) in that they temporarily store frequently-requested content. Requests for content already stored in the cache can be served from local storage instead of requiring the data to be transmitted over expensive network links.

Two types of P2P caches exist: non-transparent P2P caches and transparent P2P caches. A non-transparent cache appears as a super peer; it explicitly peers with other P2P clients. For a transparent cache, once a P2P cache is established, the network will transparently redirect P2P traffic to the cache, which either serves the file directly or passes the request on to a remote P2P user and simultaneously caches that data. Transparency is typically implemented using Deep Packet Inspection (DPI). DPI products identify and pass P2P packets to the P2P caching system so it can cache the traffic and accelerate it.

To enable operation with existing P2P software, P2P caches directly support P2P application protocols. A large number of P2P protocols are used by P2P software, and hence are supported by caches, leading to higher complexity. Additionally, these protocols evolve over time, and new protocols are introduced.

4.12.1. Applicability to DECADE

An example of in-network storage for P2P systems. However, unlike DECADE, the existence and operation of the storage is totally transparent to the end user.

4.12.2. Transparent P2P Caches

4.12.2.1. Data Access Interface

Data Access Interface allows P2P content to be cached (stored) and supplied (retrieved) locally such that network traffic is reduced, but it is transparent to P2P users, and P2P users implicitly use the data-access interface (in the form of their native P2P application protocol) to store or retrieve content.

4.12.2.2. Data Management Operations

Not provided.

4.12.2.3. Data Search Capability

Not provided.

4.12.2.4. Access Control Authorization

Access control method is typically public-restricted (to any client which is part of the P2P channel or swarm).

4.12.2.5. Resource Control Interface

Not provided.

4.12.2.6. Discovery Mechanism

Use of DPI means no discovery mechanism is provided to P2P users, it is transparent to P2P users. Since DPI is used to recognize P2P applications' private protocols, P2P Cache implementations must be updated as new applications are added and existing protocols evolve.

4.12.2.7. Storage Mode

Object-based. Chunks (typically, the unit of transfer amongst P2P clients) of content are stored in the cache.

4.12.3. Non-transparent P2P Caches

4.12.3.1. Data Access Interface

Data Access Interface allows P2P content to be cached (stored) and supplied (retrieved) locally such that network traffic is reduced. P2P users implicitly store and retrieve from the cache using the P2P application's native protocol.

4.12.3.2. Data Management Operations

Not provided.

4.12.3.3. Data Search Capability

Not provided.

4.12.3.4. Access Control Authorization

Access control method is typically public-restricted (to any client which is part of the P2P channel or swarm)

4.12.3.5. Resource Control Interface

Not provided.

4.12.3.6. Discovery Mechanism

A cache pretends to be normal peers to join the P2P overlay network. Other P2P users can find these cache nodes through overlay routing mechanism, just looking to them as normal neighbor nodes.

4.12.3.7. Storage Mode

Object-based. Chunks (typically, the unit of transfer amongst P2P clients) of content are stored in the cache.

4.13. Usenet

Usenet is a distributed Internet based discussion (message) system. The Usenet messages are arranged as a set of "newsgroups" that are classified hierarchically by subject. Usenet information is distributed and stored among a large conglomeration of servers that store and forward messages to one another in so called news feeds. Individual users may read messages from and post messages to a local news server typically operated by an ISP. This local server communicates with other servers and exchanges articles with them. In this fashion, the message is copied from server to server and eventually reaches every server in the network [25].

Traditional Usenet as described above operates as a P2P network between the servers, and in a client-server architecture between the user and their local news server. The user requires a Usenet client to be installed on their computer and a Usenet server account (through their ISP). However, with the rise of web browsers the Usenet architecture is evolving to be web based. The most popular example of this is Google Groups where Google hosts all the

newsgroups and client access is via a standard HTTP based web browser [26].

4.13.1. Applicability to DECADE

A historically very important and widely used (deployed) example of in-network storage in the Internet. The use of this system is rapidly declining but efforts have been made to preserve the stored content for historical purposes.

4.13.2. Data Access Interface

Users can read and post (store) messages.

4.13.3. Data Management Operations

Users sometimes have limited ability to delete messages that they previously posted.

4.13.4. Data Search Capability

Traditionally, users could manually search through the newsgroups as they are classified hierarchically by subject. In the newer web based systems there is also automatic search capability based on key word matches.

4.13.5. Access Control Authorization

Access control method is either public-unrestricted or private (to client members of that newsgroup).

4.13.6. Resource Control Interface

Not provided.

4.13.7. Discovery Mechanism

Usually by manually logging on to the Usenet account. DNS may be used to resolve hostnames to their corresponding addresses.

4.13.8. Storage Mode

File system. Messages are usually stored as files which are then organized hierarchically by subject into newsgroups.

4.14. Web Cache

Web cache [27] has been widely deployed by many ISPs to reduce bandwidth consumption and web access latency since the late 1990s. A web cache can cache the web documents (e.g., HTML pages, images) between server and client to reduce bandwidth usage, server load, and perceived lag. A web cache server is typically shared by many clients, and stores copies of documents passing through it; subsequent requests may be satisfied from the cache if certain conditions are met.

Another form of cache is a client-side cache, typically implemented in web browsers. A client side cache can keep a local copy of all pages recently displayed by browser, and when the user returns to one of these web pages, the local cached copy is reused.

A related protocol for P2P applications to use web cache is HPTP (HTTP based Peer to Peer) [28]. It proposes to share chunks of P2P files/streams using HTTP protocol with cache-control headers.

4.14.1. Applicability to DECADE

Very widely used (deployed) example of in-network storage for the key Internet application of Web browsing. The existence and operation of the storage is transparent to the end user in most cases. The content caching time is controlled by Time To Live parameters associated with the original content. The principle of web caching is to speed up web page reading by using (the same) content previously requested by a preceding user to service a new user.

4.14.2. Data Access Interface

Users explicitly read from a web cache by making requests, but they cannot explicitly write data into it. Data is implicitly stored into the web cache by requesting content that is not already cached and meets policy restrictions of the cache provider.

4.14.3. Data Management Operations

Not provided.

4.14.4. Data Search Capability

Not provided.

4.14.5. Access Control Authorization

Access control method for clients is public-unrestricted. It is important to note that if content is authenticated or encrypted (e.g., HTTPS, SSL) it will not be cached. Also if the content is flagged as private (vs public) at the HTTP level by the origin server it will not be cached.

4.14.6. Resource Control Interface

Not provided.

4.14.7. Discovery Mechanism

Web Caches can be transparently deployed between Web Server and Web Clients, employing DPI for discovery. Alternatively, web caches could be explicitly discovered by clients using techniques such as DNS or manual configuration.

4.14.8. Storage Mode

Object based. Web content is keyed within the cache by HTTP Request fields, such as Method, URI, and Headers.

4.15. Observations Regarding In-Network Storage Systems

This following observations about the surveyed In-Network storage systems are made in the context of DECADE as defined by [1].

The majority of the surveyed systems were designed for client-server architectures and do not support P2P. However, there are some important exceptions, especially for some of the newer technologies such as BranchCache and P2P Cache which do support a P2P mode.

The P2P cache systems are interesting since they do not require changes to P2P applications themselves. However, this is also a limitation in that they are required to support each application protocol.

Many of the surveyed systems were designed for caching as opposed to long term network storage. Thus during DECADE protocol design it should be carefully considered if a caching mode should be supported in addition to a long term network storage mode. There is typically a trade-off between providing a caching mode and long-term (and usually also reliable) storage with regards to some performance metrics. Note that [1] identifies issues with classical caching from a DECADE perspective such as the fact that P2P caches typically do not allow users to explicitly control content stored in the cache.

Certain components of the surveyed systems are outside of the scope of DECADE. For example, a protocol used for searching across multiple DECADE servers is out of scope. However, applications may still be able to implement such functionality if DECADE exposes the appropriate primitives. This has the benefit of keeping the core in-network storage systems simple, while permitting diverse applications to design mechanisms that meet their own requirements.

Today, most in-network storage systems follow some variant of the authorization model of public-unrestricted, public-restricted, and private. For DECADE, we may need to evolve the authorization model to support a resource owner (e.g., end user) authorization, in addition to the network authorization.

5. Storage And Other Related Protocols

This section surveys existing storage and other related protocols, as well as comments on the usage of these protocols to satisfy DECADE's use cases. The surveyed protocols are listed alphabetically.

5.1. HTTP

HTTP [29] is a key protocol for the World Wide Web. It is a stateless client-server protocol that allows applications to be designed using the REST model. HTTP is often associated with downloading (reading) content from web servers to web browsers, but it also has support for uploading (writing) of content to web servers. It has been used as the underlying protocol for other protocols such as WebDAV.

HTTP is used in some of the most popular in-network storage systems surveyed previously including CDNs, Photo Sharing, and Web Cache. Usage of HTTP by a storage protocol implies that no extra SW is required in the client (i.e., web based client) as all standard Web browsers are based on HTTP.

5.1.1. Data Access Interface

Basic read and write operations are supported (using HTTP GET, PUT and POST methods).

5.1.2. Data Management Operations

Not provided.

5.1.3. Data Search Capability

Not provided

5.1.4. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private.

The majority of web pages are public-unrestricted in terms of reading but do not allow any uploading of content. In-network storage systems range from private or public-unrestricted for Photo Sharing described in Section 4.11.5 to public-unrestricted for Web Caching described in Section 4.14.5.

5.1.5. Resource Control Interface

Not provided.

5.1.6. Discovery Mechanism

Manual configuration is typically used. Clients typically address HTTP servers by providing a hostname, which is resolved to an address using DNS.

5.1.7. Storage Mode

HTTP is a protocol, it thus does not define a Storage Mode. However, a non-collection resource can typically be thought of as a "file"). These files may be organized into collections, which typically map on to the HTTP Path hierarchy, creating the illusion of a file system.

5.1.8. Comments

HTTP is based on a client-server architecture and thus is not directly applicable for the DECADE focus on P2P. Also, HTTP offers only a rudimentary toolset for storage operations compared to some of the other storage protocols.

5.2. iSCSI

Small Computer System Interface (SCSI) is a set of protocols enabling communication with storage devices such as disk drives and tapes; internet SCSI (iSCSI) [30] is a protocol enabling SCSI commands to be sent over TCP. As in SCSI, iSCSI allows an Initiator to send commands to a Target. These commands operate on the device level as opposed to individual data objects stored on the device.

5.2.1. Data Access Interface

Read and write commands indicate which data is to be read or written by specifying the offset (using Logical Block Addressing) into the storage device. The size of data to be read or written is an additional parameter in the command.

5.2.2. Data Management Operations

Since commands operate at the device level, management operations are different than with traditional file systems. Management commands for SCSI/iSCSI including explicit device control such as starting and stopping the device and formatting the device.

5.2.3. Data Search Capability

SCSI/iSCSI does not provide the ability to search for particular data within a device. Note that such capabilities can be implemented outside of iSCSI.

5.2.4. Access Control Authorization

With respect to access to devices, the access control method is private. iSCSI uses CHAP [31] to authenticate initiators and targets when accessing storage devices. However, since SCSI/iSCSI operates at the device level, neither authentication nor authorization are provided for individual data objects. Note that such capabilities can be implemented outside of iSCSI.

5.2.5. Resource Control Interface

Not provided.

5.2.6. Discovery Mechanism

Manual configuration may be used. An alternative is the internet Storage Name Service (iSNS) [32] provides the ability to discover available storage resources.

5.2.7. Storage Mode

As a protocol, iSCSI does not explicitly have a storage mode. However, it provides block-based access to clients. SCSI/iSCSI provides an Initiator block-level access to the storage device.

5.3. NFS

The Network File System is designed to allow users to access files over a network in a manner similar to how local storage is accessed. NFS is typically used in local area network or enterprise settings, though changes made in later versions of NFS make it easier to operate over the Internet.

5.3.1. Data Access Interface

Traditional file-system operations such as read, write, and update (overwrite) are provided. Locking is provided to support concurrent access by multiple clients.

5.3.2. Data Management Operations

Traditional file-system operations such as move and delete are provided.

5.3.3. Data Search Capability

User has the ability to list contents of directories to find filenames matching desired criteria.

5.3.4. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private. For example, files and directories can be protected using read, write, and execute permissions for the files owner, group, and the public (others). Also, NFSv4.1 has a rich ACL model allowing a list of Access Control Entries (ACEs) to be configured for each file or directory. The ACEs can specify per-user read/write access to file data, file/directory attributes, creation/deletion of files in a directory, etc.

5.3.5. Resource Control Interface

While disk space quotas can be configured, it typically limits the total amount of storage allocated to a particular user. User control of bandwidth and connections used by remote peers is not provided.

5.3.6. Discovery Mechanism

Manual configuration is typically used. Clients address NFS servers by providing a hostname and a directory that should be mounted. DNS may be used lookup an address for the provided hostname.

5.3.7. Storage Mode

As a protocol, there is no defined internal storage mode. However, implementations typically use the underlying filesystem storage. Note that extensions have been defined for alternate storage modes (e.g., block-based [34] and object-based [35]).

5.3.8. Comments

The efficiency and scalability of the NFS access control method is a concern in the context of DECADE. In particular, Section 6.2.1 of [33] states that:

> Only ACEs that have a "who" that matches the requester are considered.

Thus, in the context of DECADE, to specify per-peer access control policies for an object, a client would need to explicitly configure the ACL for the object for each individual peer. A concern with this approach is scalability when a client's peers may change frequently and ACLs for many small objects need to be updated constantly during participation in a swarm.

Note that NFS v4.1's usage of RPCSEC_GSS provides support for multiple security mechanisms. Kerberos V5 is required, but others such as X.509 certificates are also supported by way of GSS-API. Note, however, that NFSv4.1's usage of such security mechanisms is limited to linking a requesting user to a particular account maintained by the NFS server.

5.4. OAuth

OAuth [36] is a protocol that enriches the traditional client-server authentication model for web resources. In particular, OAuth distinguishes the "client" from the "resource owner", thus enabling a resource owner to authorize a particular client for access (e.g., for a particular lifetime) to private resources.

We include OAuth in this survey so that its authentication model can be evaluated in the context of DECADE. OAuth itself, however, is not a network storage protocol.

5.4.1. Data Access Interface

Not provided.

5.4.2. Data Management Operations

Not provided.

5.4.3. Data Search Capability

Not provided.

5.4.4. Access Control Authorization

Not provided. While similar in spirit to the WebDAV ticketing extensions [37], OAuth instead uses the following process: (1) a client constructs a delegation request, (2) the client forwards the request to the resource owner for authorization, (3) the resource owner authorizes the request, and finally (4) a callback is made to the client indicating that its request has been authorized.

Once the process is complete, the client has a set of token credentials that grant it access to the protected resource. The token credentials may have an expiration time, and they can also be revoked by the resource owner at any time.

5.4.5. Resource Control Interface

Not provided.

5.4.6. Discovery Mechanism

Not provided.

5.4.7. Storage Mode

Not provided.

5.4.8. Comments

The ticketing mechanism requires server involvement and the discussion relating to WebDAV's proposed ticketing mechanism (see Section 5.5.8) applies here as well.

5.5. WebDAV

WebDAV [38] is a protocol designed for Web content authoring. It is developed as an extension to HTTP described in Section 5.1, meaning it can be simpler to integrate into existing software. WebDAV supports traditional operations for reading/writing from storage, as well as other constructs such as locking and collections which are important when multiple users collaborate to author or edit a set of

documents.

5.5.1. Data Access Interface

Traditional read and write operations are supported (using HTTP GET and PUT methods, respectively). Locking is provided to ease concurrent access by multiple clients.

5.5.2. Data Management Operations

WebDAV supports traditional file-system operations such as move, delete and copy. Objects are organized into collections, and these operations can also be performed on collections. WebDAV also allows objects to have user-defined properties.

5.5.3. Data Search Capability

User has the ability to list contents of collections to find objects matching desired criteria. A SEARCH extension [39] has also been specified allowing listing of objects matching client-defined criteria.

5.5.4. Access Control Authorization

All methods of access control for clients are supported: public-unrestricted, public-restricted and private.

For example, an ACL extension [40] is provided for WebDAV. ACLs allow both user- and group-based access control policies (relating to reading, writing, properties, locking, etc) to be defined for objects and collections.

A ticketing extension [37] has also been proposed, but has not progressed beyond an Internet Draft. This extension allows a client to request the WebDAV server to create a "ticket" (e.g., for reading an object) that can be distributed to other clients. Tickets may be given expiration times, or may only allow for a fixed number of uses. The proposed extension requires the server to generate tickets and maintain state for outstanding tickets.

5.5.5. Resource Control Interface

An extension [41] allows disk space quotas to be configured for Collections. The extension also allows WebDAV clients to query current disk space usage. User control of bandwidth and connections used by remote peers is not provided.

5.5.6. Discovery Mechanism

Manual configuration is typically used. Clients address WebDAV servers by providing a hostname, which can be resolved to an address using DNS.

5.5.7. Storage Mode

Though no storage mode is explicitly defined, WebDAV can be thought of as providing file-based storage to a client. A non-collection resource can typically be thought of as a "file". Files may be organized into collections, which typically map on to the HTTP Path hierarchy.

5.5.8. Comments

The efficiency and scalability of the WebDAV access control method is a concern in the context of DECADE, for similar reasons as stated in Section 5.3.8 for NFS. The proposed WebDAV ticketing extension partially alleviates this concern, but the particular technique may need further evaluation before being applied to DECADE. In particular, since DECADE clients may continuously upload/download a large number of small-size objects, and a single DECADE server may need to scale to many concurrent DECADE clients, requiring the server to maintain ticket state and generate tickets may not be the best design choice. Server-generated tickets can also increase latency for data transport operations depending on the message flow used by DECADE.

5.6. Observations Regarding Storage and Related Protocols

This following observations about the surveyed storage and related protocols are made in the context of DECADE as defined by [1].

All of the surveyed protocols were primarily designed for client-server architectures and not for P2P. However, it is conceivable that some of the protocols could be adapted to work in a P2P architecture.

Several popular in-network storage systems today use HTTP as their key protocol even though it is not classically considered as a storage protocol. HTTP is a stateless protocol that is used to design RESTful applications. HTTP is a well supported and widely implemented protocol which can provide important insights for DECADE.

The majority of the surveyed protocols do not support low latency access for applications such as live streaming. This was one of the key general requirements for DECADE.

The majority of the surveyed protocols do not support any form of resource control interface. Resource control is required for users to manage the resources on in-network storage that can be used by other peers, e.g., the bandwidth or connections. Resource control is a key capability required for DECADE.

Nearly all surveyed protocols did however support the following capabilities which are required for DECADE: user ability to read/ write content; some form of access control; some form of error indication; and ability to traverse firewalls and NATs.

6. Conclusions

Though there have been many successful in-network storage systems, they have been designed for use cases different from those defined in DECADE. For example, many of the surveyed in-network storage systems and protocols were designed for client-server architectures and not P2P. No surveyed system or protocol has the functionality and features to fully meet the set of requirements defined for DECADE. DECADE aims to provide a standard protocol for P2P applications and content provider to access and control in-network storage, resulting in increased network efficiency while retaining control over content shared with peers. Additionally, defining a standard protocol can reduce complexity of in-network storage since multiple P2P application protocols no longer need to be implemented by in-network storage systems.

7. Security Considerations

This draft is a survey of existing in-network storage systems, and does not introduce any security considerations beyond those of the surveyed systems.

For more information on security considerations of DECADE, see [1].

8. IANA Considerations

This document does not have any IANA Considerations.

9. Contributors

The editors would like to thank the following people for contributing to the development of this document:

- ZhiHui Lu

- Borje Ohlman

- Pang Tao

- Lucy Yong

- Juan Carlos Zuniga

10. Acknowledgments

The editors would like to thank the following people for providing valuable comments to various versions of this document: David Bryan, Tao Mao, Haibin Song, Ove Strandberg, Yu-Shun Wang, Richard Woundy, Yunfei Zhang, and Ning Zong.

11. Informative References

[1] Song, H., Zong, N., Yang, Y., and R. Alimi, "DECoupled Application Data Enroute (DECADE) Problem Statement", draft-ietf-decade-problem-statement-03 (work in progress), March 2011.

[2] Storage Search, "Flash Memory vs. Hard Disk Drives - Which Will Win?", http://www.storagesearch.com/semico-art1.html.

[3] Matt's Computer Trends, "Flash and Disk Trends", http://www.mattscomputertrends.com/flashdiskcomparo.html.

[4] Yingjie, G., Bryan, D., Yang, Y., and R. Alimi, "DECADE Requirements", draft-ietf-decade-reqs-03 (work in progress), July 2011.

[5] Amazon, "Amazon Simple Storage Service (Amazon S3)", http://aws.amazon.com/s3/.

[6] Microsoft Corporation, "Windows Azure Blob - Programming Blob Storage".

[7] Google, "Google Storage for Developers", http://code.google.com/apis/storage.

[8] Dropbox, "Dropbox Features", http://www.dropbox.com/features.

[9] Microsoft Corporation, "BranchCache",

http://technet.microsoft.com/en-us/network/dd425028.aspx.

[10] S. Paul, R. Yates, D. Raychaudhuri, J. Kurose., "The Cache-and-Forward Network Architecture for Efficient Mobile Content Delivery Services in the Future Internet", In Innovations in NGN: Future Network and Services, 2008.

[11] SNIA, "Cloud Data Management Interface", http://www.snia.org/cdmi.

[12] Pathan, A.K., Buyya, R., "A Taxonomy and Survey of Content Delivery Networks", In Grid Computing and Distributed Systems Laboratory in University of Melbourne, Technology Report, Feb. 2007.

[13] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

[14] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

[15] Named Data Networking, "Named Data Networking Home Page", http://www.named-data.net/.

[16] Named Data Networking, "Named Data Networking Project Proposal", http://www.named-data.net/ndn-proj.pdf.

[17] Network of Information., "Network of Information Overview", http://www.netinf.org/home/overview/.

[18] A. Anand, V. Sekar, A. Akella., "SmartRE: An Architecture for Coordinated Network-wide Redundancy Elimination", In SIGCOMM 2009.

[19] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz., "Pond: the OceanStore Prototype", In FAST 2003.

[20] Kodak, "Kodak Gallery Home Page", http://www.kodakgallery.com/gallery/welcome.jsp.

[21] Wikipedia, "Kodak Gallery", http://en.wikipedia.org/wiki/Kodak_Gallery.

[22] Flickr, "Flickr Home Page", http://www.flickr.com.

[23] ImageShack, "ImageShack Home Page", http://imageshack.us.

[24] Tumblr, "Tumblr Home Page", http://www.tumblr.com.

[25] Wikipedia, "Usenet", http://en.wikipedia.org/wiki/Usenet.

[26] Google, "Google Groups", http://groups.google.com.

[27] Geoff Huston, Telstra., "Web Caching", In The Internet Protocol Journal Volume 2, No. 3.

[28] G. Shen, Y. Wang, Y. Xiong, B.Y. Zhao, Z.-L. Zhang, "HPTP: Relieving the tension between isps and p2p", In 6th International workshop on Peer-To-Peer Systems (IPTPS2007).

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

[30] Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M., and E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)", RFC 3720, April 2004.

[31] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[32] Tseng, J., Gibbons, K., Travostino, F., Du Laney, C., and J. Souza, "Internet Storage Name Service (iSNS)", RFC 4171, September 2005.

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

[34] Black, D., Fridella, S., and J. Glasgow, "Parallel NFS (pNFS) Block/Volume Layout", RFC 5663, January 2010.

[35] Halevy, B., Welch, B., and J. Zelenka, "Object-Based Parallel NFS (pNFS) Operations", RFC 5664, January 2010.

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

[37] Ito, K., "Ticket-Based Access Control Extension to WebDAV", draft-ito-dav-ticket-00 (work in progress), October 2001.

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

[39] Reschke, J., Reddy, S., Davis, J., and A. Babich, "Web

Distributed Authoring and Versioning (WebDAV) SEARCH", RFC 5323, November 2008.

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

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

Authors' Addresses

Richard Alimi (editor)
Google

Email: ralimi@google.com

Akbar Rahman (editor)
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

Yang Richard Yang (editor)
Yale University

Email: yry@cs.yale.edu