Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 14, 2014

N. Zong, Ed.
X. Chen
Z. Huang
Huawei Technologies
L. Chen
HP Labs
H. Liu
Yale University

August 13, 2013

Integration Examples of DECADE System
draft-zong-integration-example-01

Abstract

Decoupled Application Data Enroute (DECADE) system is an in-network storage infrastructure which was discussed in IETF. This document presents two detailed examples of how to integrate such in-network storage infrastructure into peer-to-peer (P2P) applications to achieve more efficient content distribution, and Application Layer Traffic Optimization (ALTO) system to build a content distribution platform for Content Providers (CPs).

Status of This Memo

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

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

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

This Internet-Draft will expire on February 14, 2014.

Table of Contents

1. Introduction

Decoupled Application Data Enroute (DECADE) system is an in-network storage infrastructure which was discussed in IETF. We implemented such in-network storage infrastructure to simulate DECADE system including DECADE servers, DECADE clients and DECADE protocols [I-D .alimi-protocol]. Therefore, in the whole draft, we use the terms of in-network storage (INS) system, INS server, INS client, INS operations, etc.

This draft introduces some examples of integrating INS system with existing applications. In our example systems, the core components include INS server and INS-enabled application client. An INS server stores data inside the network, and thereafter manages both the stored data and access to that data. An INS-enabled application client including INS client and native application client uses a set of Application Programming Interfaces (APIs) to enable native application client to utilize INS operations such as data get, data put, storage status query, etc.

This draft presents two detailed examples of how to integrate INS system into peer-to-peer (P2P) applications, i.e. live streaming and file sharing, as well as an example integration of Application Layer

Traffic Optimization (ALTO) [I-D.ietf-alto-protocol] and INS system to support file distribution. We show how to extend native P2P applications by designing the INS-enabled P2P clients and describing the corresponding flows of INS-enabled data transmission. Then we introduce the functional architecture and working flows of integrated ALTO and INS system for file distribution of Content Providers (CPs). Finally we illustrate the performance gain to P2P applications and more efficient content distribution by effectively leveraging the INS system.

Please note that the P2P applications mentioned in this draft only represent some cases out of a large number of P2P applications, while the INS system itself can support a variety of other applications. Moreover, the set of APIs used in our integration examples is an experimental implementation, which is not standard and still under development. The INS system described in this draft is only a preliminary functional set of in-network storage infrastructure for applications. It is designed to test the pros and cons of INS system utilized by P2P applications and verify the feasibility of utilizing INS system to support content distribution. We hope our examples would be useful for further standard protocol design, rather than to present a solution for standardization purpose.

2. Terminology

The following terms will be used in this document.

2.1. Native Application Client

A client running original application operations including control and data messages defined by applications.

2.2. INS Server

A server to simulate DECADE server defined in [I-D.alimi-protocol].

2.3. INS Client

A client to simulate DECADE client defined in [I-D.alimi-protocol].

2.4. INS Operations

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

2.5. INS System

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

2.6. INS Client API

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

2.7. INS-enabled Application Client

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

2.8. INS Service Provider

An INS service provider deploys INS system and provides INS service to applications/end users. It can be Internet Service Provider (ISP) or other parties.

2.9. INS Portal

A functional entity operated by INS service provider to offer applications/end users a point to access (e.g. upload, download) files stored in INS servers.

3. INS Client API

In order to simplify the integration of INS system with P2P applications, we provide INS client API to native P2P clients for accomplishing INS operations such as data get, data put, etc. On top of the INS client API, a native P2P client can develop its own application specific control and data distribution flows.

We currently developed the following five basic interfaces.

o Generate_Token: Generate an authorization token. An authorization token is usually generated by an entity that is trusted by an INS client which is sharing its data and passed to the other INS clients for data access control. Please see [I-D.alimi-protocol] for more details.

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

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

o Delete_Object: Delete a data object in an INS server explicitly with an authorization token. Note that a data object can also be deleted implicitly by setting a Time-To-Live (TTL) value.

o Status_Query: Query current status of an application itself, including listing stored data objects, resource (e.g. storage space) usage, etc.

4. Integration of P2P File Sharing and INS System

We integrate an INS client into Vuze - a BitTorrent based file sharing application [VzApp].

4.1. Integration Architecture

The architecture of the integration of Vuze and INS system is shown in Figure 1. An INS-enabled Vuze client uses INS client to communicate with INS server and transmit data between itself and INS server. It is also compatible with original Vuze signaling messages such as peer discovery, data availability announcement, etc. Note that the same architecture applies to the other example of integration of P2P live streaming and INS system.

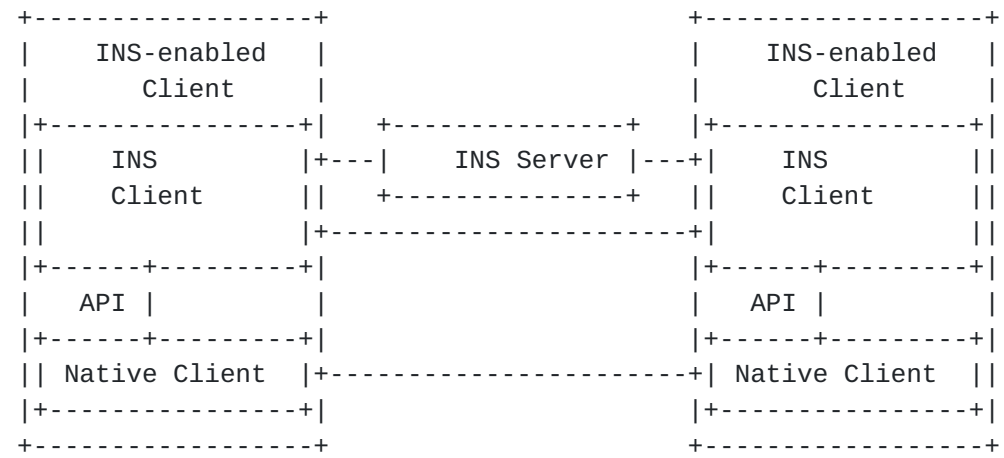
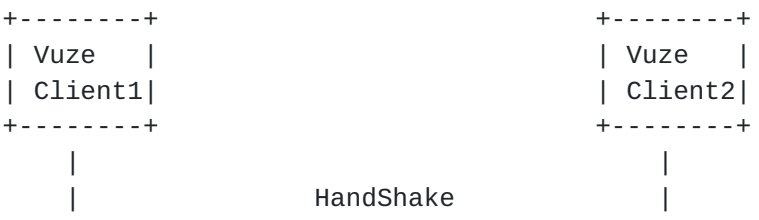


Figure 1

4.1.1. Message Flow

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



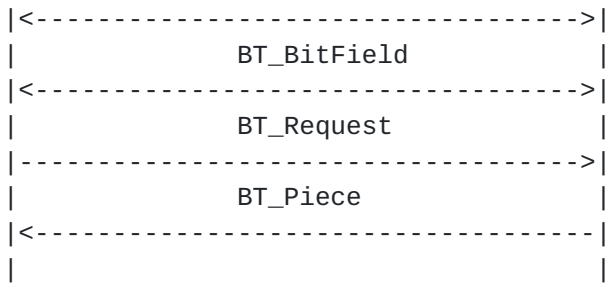


Figure 2

In the above diagram, one can see that the key messages for data sharing in native Vuze are "BT_BitField", "BT_Request" and "BT_Piece". Vuze client1 and client2 exchange "BT_BitField" messages to announce the available data objects to each other. If Vuze client1 wants to get certain data object from client2, it sends a "BT_Request" message to client2. Vuze client2 then return the requested data object to client1 by a "BT_Piece" message. Please refer to [VzMsg] for the detailed description of Vuze messages.

As shown in the below diagram, in the integration of Vuze and INS system, INS client inserts itself into the Vuze client by intercepting certain Vuze messages, and adjusting their handling to send/receive data using the INS operations instead.

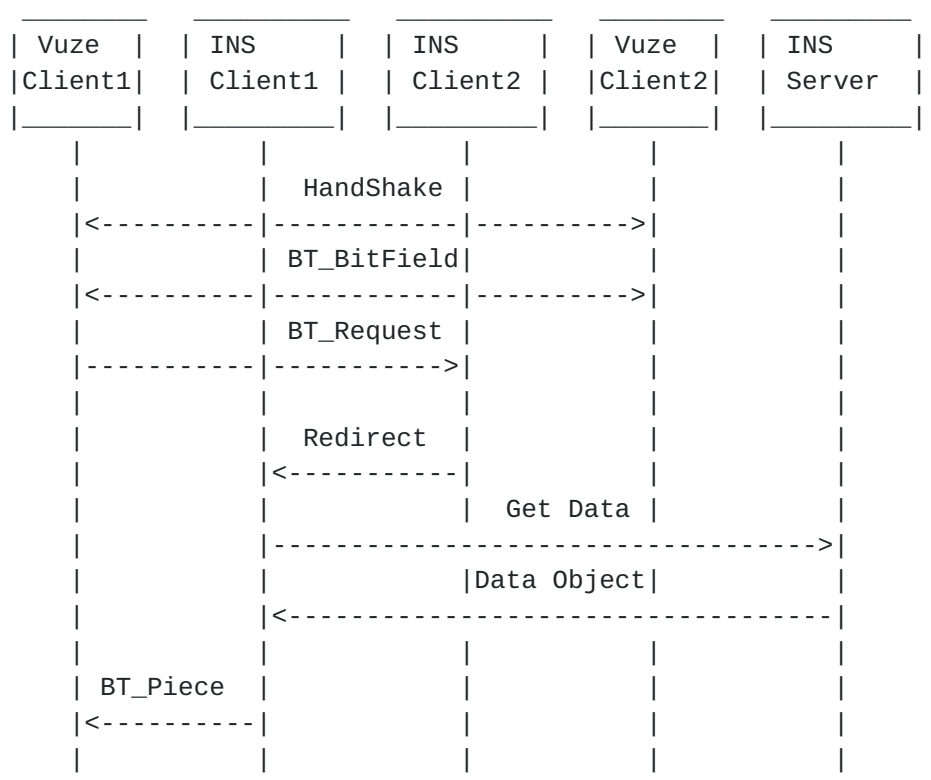


Figure 3

o Vuze client1 sends a "BT_Request" message to Vuze client2 to request a data object as usual.

o INS client2 embedded in Vuze client2 intercepts the incoming "BT_Request" message and then replies with a "Redirect" message which includes INS server's address and authorization token.

o INS client1 receives the "Redirect" message and then sends an INS message "Get Data" to the INS server to request the data object.

o INS server receives the "Get Data" message and sends the requested data object back to INS client1 after the token check.

o INS client1 encapsulates the received data object into a "BT_Piece" message and sends to Vuze client1.

In this example, the file to be shared is divided into many objects, with each object being named as "filename_author_partn" where author is the original author of the file or the user who uploads the file, n is the sequence number of the object.

4.2. Concluding Remarks

In this example, we feel that the INS system can effectively improve the file sharing efficiency due to following reasons: 1) utilizing in-network storage as the data location of the peer will achieve statistical multiplexing gain of the data sharing; 2) shorter data delivery path based on in-network storage could not only improve the application performance, but avoid the potential bottleneck in the ISP network.

5. Integration of P2P Live Streaming and INS System

We integrate an INS client into a P2P live streaming application.

5.1. Integration Architecture

The architecture of the integration of P2P live streaming application and INS system is shown in Figure 1. An INS-enabled P2P live streaming client uses INS client to communicate with INS server and transmit data between itself and INS server.

5.1.1. Data Access Messages

INS client API is called whenever an INS-enabled P2P live streaming client wants to get data objects from (or put data objects into) the INS server. Each data object transferred between the application client and the INS server should go through the INS client. Each data object can be a variable-sized block to cater to different application requirements (e.g. latency and throughput).

We use the hash of a data object's content for the name of the data object. The name of a data object is generated and distributed by the source streaming server in this example.

5.1.2. Control Messages

We used a lab-based P2P live streaming system for research purpose only. The basic control messages between the native P2P live streaming clients are similar to Vuze control protocols in the sense that the data piece information is exchanged between the peers. The INS-enabled P2P live streaming client adds an additional control message for authorization token distribution, as shown as the line between the INS clients in Figure 1. In this example, the authorization token is generated by the INS client that is sharing its data. By exchanging the authorization tokens, the application clients can retrieve the data objects from the INS servers.

5.2. Design Considerations

One essential objective of the integration is to improve the performance of P2P live streaming application. In order to achieve such goal, we have some important design considerations that would be helpful to the future work of protocol development.

5.2.1. Improve Efficiency for Each Connection

In a native P2P system, a peer can establish tens or hundreds of concurrent connections with other peers. On the other hand, it may be expensive for an INS server to maintain many connections for a large number of INS clients. Typically, each INS server may only allocate and maintain M connections (in our examples, M=1) with each INS client at a time. Therefore, we have the following design considerations to improve the efficiency for each connection between INS server and INS client to achieve satisfying data downloading performance.

o Batch Request: In order to fully utilize the connection bandwidth of INS server and reduce the overhead, an application client may request a batch of data objects in a single request.

o Larger Data Object: Data object size in existing P2P live streaming application may be small and thus incur large control overhead and low transport utilization. A larger data object may be needed to more efficiently utilize the data connection between INS server and INS client.

5.2.2. Reduce Control Latency

In a native P2P system, a serving peer sends data objects to the requesting peer directly. Nevertheless, in an INS system, the serving client typically only replies with an authorization token to the requesting client, and then the requesting client uses this token to fetch the data objects from the INS server. This process introduces an additional control latency compared with the native P2P system. It is even more serious in latency sensitive applications such as P2P live streaming. Therefore, we need to consider how to reduce such control latency.

o Range Token: One way to reduce control latency is to use range token. An INS-enabled P2P live streaming client may piggyback a range token when announcing data availability to other peers, indicating that all available data objects are accessible by this range token. Then instead of requesting some specific data object and waiting for the response, a peer can use this range token to access all available data objects that it was permitted to access in the INS server.

6. Integration of ALTO and INS System for File Distribution

The objective of ALTO service is to give guidance to applications about which content servers to select to improve content distribution performance in an ISP-friendly way (e.g. reducing network usage within the ISP). The core component of ALTO service is called ALTO server which generates the guidance based on the ISP network information. The ALTO protocol conveys such guidance from the ALTO server to the applications. The detailed description of ALTO protocol can be found in [I-D.ietf-alto-protocol].

In this example, we integrate ALTO and INS system to build a content distribution platform for CPs.

6.1. Architecture

The integrated system allows CPs to upload files to INS servers, and guides end users to download files from the INS servers suggested by ALTO service. The architecture diagram is shown as below. Note that this diagram just shows a basic set of connections between the components. Some redirection including that the INS portal redirects end users to the INS servers can also happen between the components.

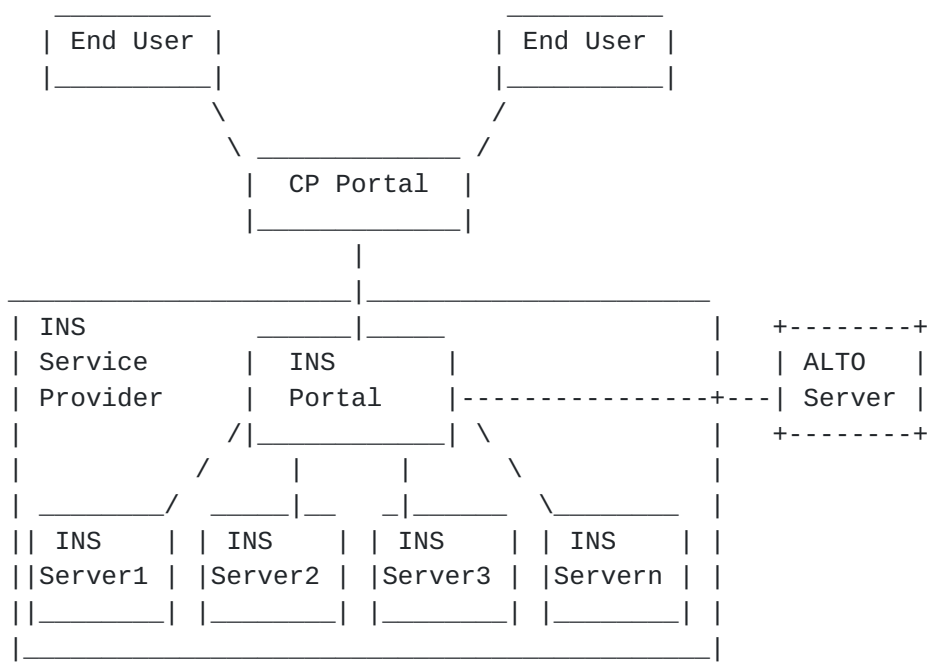


Figure 4

Four key components are defined as follow.

o INS Servers: operated by an INS service provider to store files from CPs.

o INS Portal: operated by an INS service provider to 1) upload files from CPs to the dedicated INS servers; 2) direct end users to the INS servers suggested by ALTO service to download files.

o CP Portal: operated by a CP to publish the URLs of the uploaded files for end user downloading.

o End User: End users use standard web browser with INS extensions such that INS client APIs can be called for fetching the data from INS servers.

6.1.1. CP Uploading Procedure

CP uploads the files into INS servers first, then gets the URLs of the uploaded files and publishes the URLs on the CP portal for end user downloading. The flow is shown below.

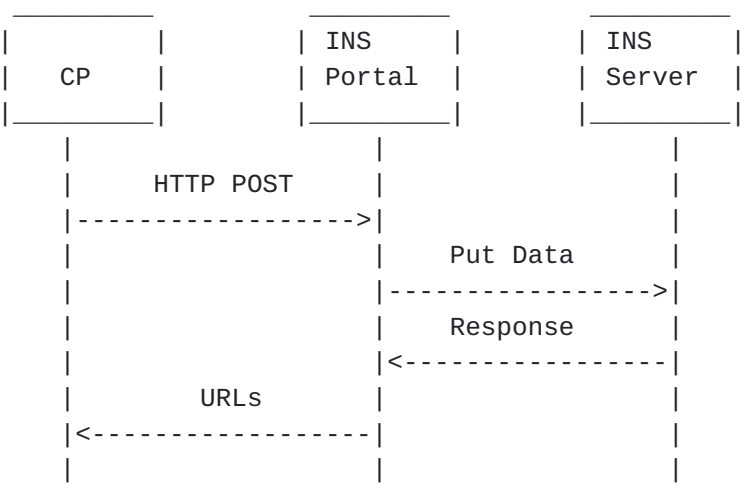


Figure 5

o CP uploads the file to the INS portal site via HTTP POST message.

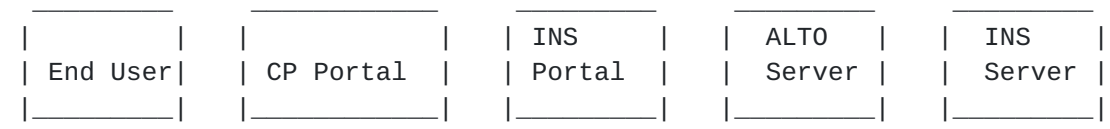
o INS portal distributes the file to the dedicated INS severs using INS message "Put Data". Note that the data distribution policies (e.g. how many copies of the data to which INS servers) can be specified by CP. The dedicated INS servers can be also decided by the INS service provider based on policies or system status (e.g. INS server load). These issues are out of the scope of this draft.

In this example, the data stored in INS server is divided into many objects, with each object being named as "filename_CPname_partn" where CPname is the name of the CP who uploads the file, n is the sequence number of the object.

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

6.1.2. End User Downloading Procedure

End users can visit the CP portal web pages and click the URLs for downloading the desired files. The flow is shown below.



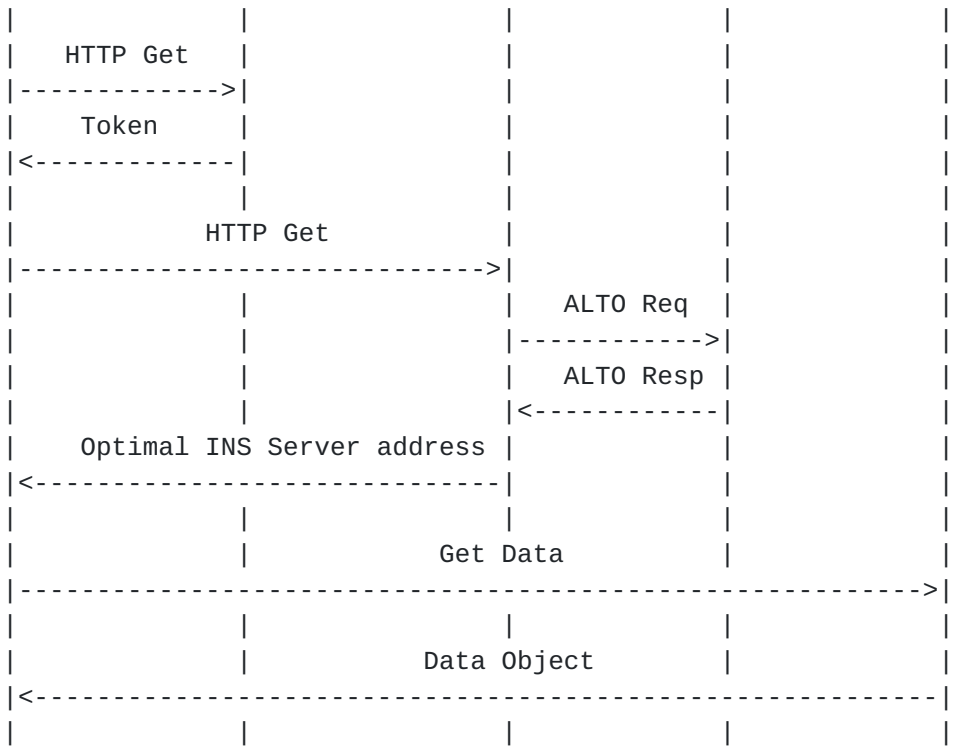


Figure 6

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

o End user clicks the hyper link, CP portal returns authorization token to the end user and redirects the end user to INS portal via HTTP Get message.

o INS portal communicates with ALTO server to get the suggested INS server storing the requested file. In this example, ALTO server just selects the INS server within the same IP subset of the end user. Please see [I-D.ietf-alto-protocol] for more details on how ALTO select content server.

o INS portal returns the INS server address suggested by ALTO service to the end user.

o End user connects to the suggested INS server to get data via INS message "Get Data" after the token check.

7. Test Environment and Settings

We conduct some tests to show the results of our integration examples. For a better performance comparison, we ran experiments

(i.e. INS integrated P2P application v.s. native P2P application) in the same environment using the same settings.

7.1. Test Settings

Our tests ran on a wide-spread area and diverse platforms, including a famous commercial platform - Amazon EC2 [EC2] and a well known test-bed - PlanetLab [PL]. The experimental settings are as follows.

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

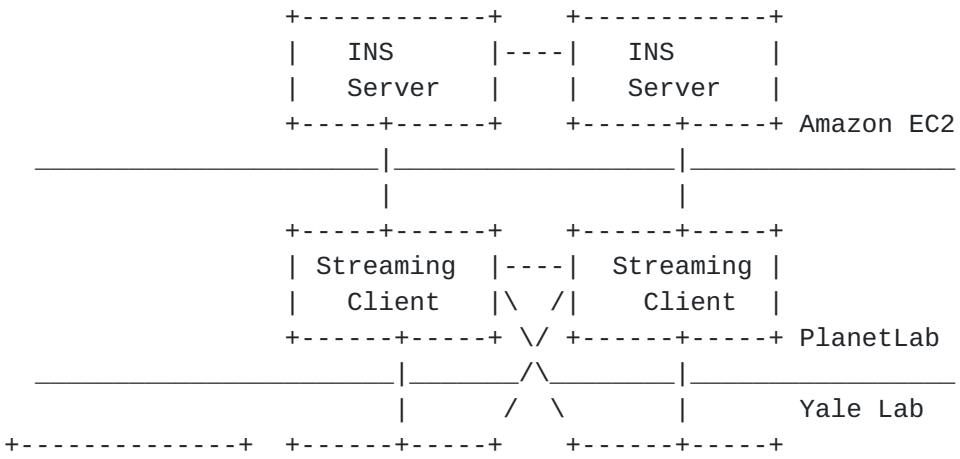
o PlanetLab: We ran our P2P live streaming clients and P2P file sharing clients (both INS-enabled and native clients) on PlanetLab on a wild-spread area.

o Flash-crowd: Flash-crowd is an important scenario in P2P live streaming system due to the live nature, i.e. a large number of users join the live channel during the startup period of the event. Therefore, we conduct experiments to test the system performance for flash-crowd in our P2P live streaming example.

o Total supply bandwidth: Total supply bandwidth is the sum of the capacity of bandwidth used to serve the streaming/file content, from both servers (including source servers and INS servers) and the P2P clients. For a fair comparison, we set the total supply bandwidth to be the same in both tests of native and INS-enabled P2P applications.

7.2. Test Environment for P2P Live Streaming Example

In the tests, we have some functional components running in different platforms, including INS servers, P2P live streaming clients (INS-enabled or native), native P2P live streaming tracker, streaming source server and test controller, as shown in below figure.



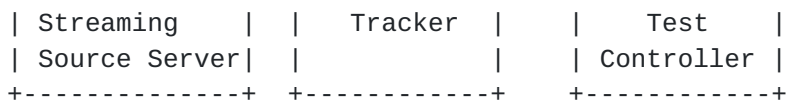


Figure 7

7.2.1. INS Server

INS servers ran on Amazon EC2.

7.2.2. P2P Live Streaming Client

Both INS-enabled and native P2P live streaming clients ran on PlanetLab. Each INS-enabled P2P live streaming client connects to the dedicated INS server. In this example, we decide which client connects to which server based on the IP address. So, it is roughly region-based and still coarse. Each INS-enabled P2P live streaming client uses its INS server to share streaming content to other peers.

7.2.3. Tracker

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

7.2.4. Streaming Source Server

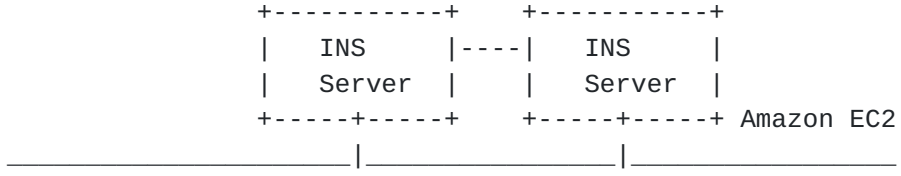
A streaming source server ran at Yale's laboratory and served both INS-enabled and native P2P live streaming clients during the test.

7.2.5. Test Controller

Test controller is a manager running at Yale's laboratory to control all machines' behaviors in both Amazon EC2 and PlanetLab during the test.

7.3. Test Environment for P2P File Sharing Example

Functional components include Vuze client (with and without INS client), INS servers, native Vuze tracker, HTTP server, PlanetLab manager and test controller, as shown in below figure.



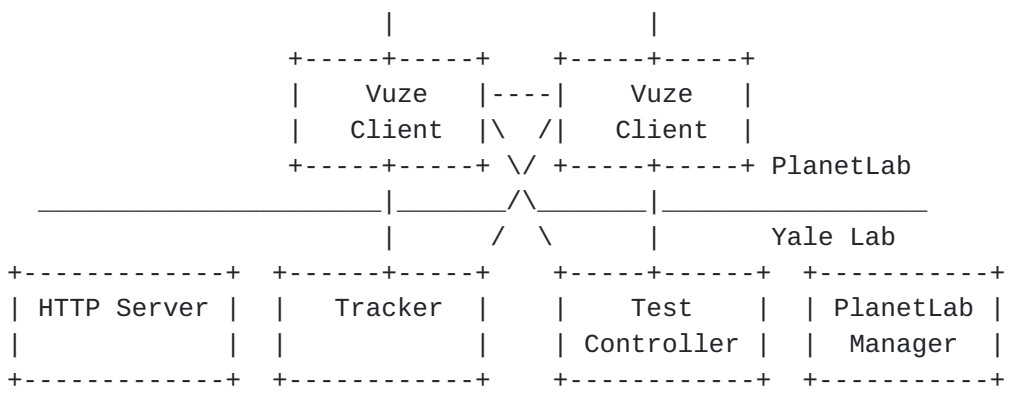


Figure 8

7.3.1. INS Server

INS servers ran on Amazon EC2.

7.3.2. Vuze Client

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

7.3.3. Tracker

Vuze software includes tracker implementation, so we didn't deploy our own tracker. Tracker ran at Yale's laboratory and was enabled when making a BitTorrent file. Tracker ran at the same Window 2003 server with the seeding client.

7.3.4. Test Controller

Similar to the test controller in P2P live streaming case, the test controller in Vuze example can also control all machines' behaviors in Amazon EC2 and PlanetLab. For example, it lists all the Vuze clients via GUI and controls them to download a specific BitTorrent file. Test controller ran at the same Window 2003 server with the seeding client.

7.3.5. HTTP Server

BitTorrent file was put in the HTTP server and the leechers retrieved the BitTorrent file from the HTTP server after receiving the downloading command from the test controller. We used Apache Tomcat for HTTP server.

7.3.6. PlanetLab Manager

PlanetLab manager is a tool developed by University of Washington. It presents a simple GUI to control PlanetLab nodes and perform common tasks such as: 1) selecting nodes for your slice; 2) choosing nodes for your experiment based on the information about the nodes; 3) reliably deploying you experiment files; 4) executing commands on every node in parallel; 5) monitoring the progress of the experiment as a whole, as well as viewing console output from the nodes.

7.4. Test Environment for Combined ALTO and INS File Distribution System

For the integration of ALTO and INS systems for supporting file distribution of CPs, we built 6 Linux virtual machines (VMs) with Fedora13 operating system. ALTO server, INS portal, CP portal and two INS servers ran on these VMs. Each VM is allocated with 4 cores from a 16-core 1Ghz CPU, and has 2GB memory space and 10GB disk space. CP uploaded files to the INS server via INS portal. End user can choose desired file through the CP portal, and download it from the optimal INS server chosen by the INS portal using ALTO service.

8. Performance Analysis

We illustrate the performance gain to P2P applications and more efficient content distribution by effectively leveraging the INS system. For the example of integrating ALTO and INS systems to support file distribution of CPs, we show the feasibility of such integration.

8.1. Performance Metrics

8.1.1. P2P Live Streaming

To measure the performance of a P2P live streaming application, we mainly employed the following four metrics.

o Startup delay: The duration from a peer joins the streaming channel to the moment it starts to play.

o Piece missed rate: The number of pieces a peer loses when playing over the total number of pieces.

o Freeze times: The number of times a peer re-buffers during playing.

o Average peer uploading rate: Average uploading bandwidth of a peer.

8.1.2. P2P File Sharing

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

o Download traffic: The total amount of traffic representing the network downlink resource usage.

o Upload traffic: The total amount of traffic representing the network uplink resource usage.

o Network resource efficiency: The ratio of P2P system download rate to the total network (downlink) bandwidth.

8.1.3. Integration of ALTO and INS System for File Distribution

We consider some common capacity metrics for content distribution system, i.e. the bandwidth usage of each INS server, and the total online users supported by each INS server.

8.2. Results and Analysis

8.2.1. P2P Live Streaming

o Startup delay: In the test, INS-enabled P2P live streaming clients startup around 35~40 seconds and some of them startup around 10 seconds. Native P2P live streaming clients startup around 110~120 seconds and less than 20% of them startup within 100 seconds.

o Piece missed rate: In the test, both INS-enabled P2P live streaming clients and native P2P live streaming clients achieved a good performance in piece missed rate. Only about 0.02% of total pieces missed in both cases.

o Freeze times: In the test, native P2P live streaming clients suffered from more freezing times than INS-enabled P2P live streaming clients by 40%.

o Average peer uploading rate: In the test, according to our settings, INS-enabled P2P live streaming clients had no data upload in their "last mile" access network, while in the native P2P live

streaming system, most peers uploaded streaming data for serving other peers. In another word, INS system can shift uploading traffic from clients' "last mile" to in-network devices, which saves a lot of expensive bandwidth on access links.

8.2.2. P2P File Sharing

The test result is illustrated in below figure. We can see that there is very few upload traffic from the INS-enabled Vuze clients, while in the native Vuze case, the upload traffic from Vuze clients is the same as the download traffic. Network resource usage is thus reduced in the "last mile" in the INS-enabled Vuze case. This result also verifies that the INS system can shift uploading traffic from clients' "last mile" to in-network devices. Note that because not all clients finish downloading process, there are different total download traffic for the independent tests, as shown in below figure.

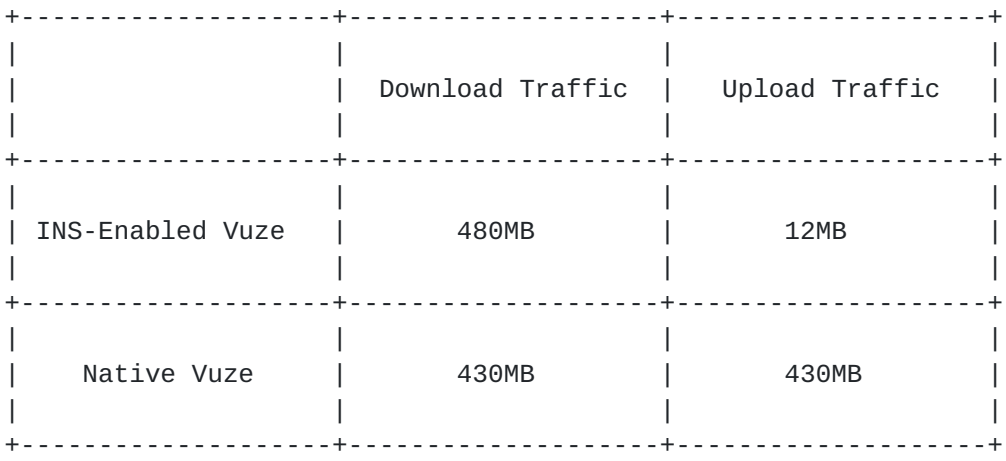


Figure 9

We also found higher network resource efficiency in the INS-enabled Vuze case where the network resource efficiency is defined as the ratio of P2P system download rate to the total network (downlink) bandwidth. The test result is that the network resource efficiency of native Vuze is 65% while that of INS-enabled Vuze is 88%. A possible reason behind the higher network resource efficiency is that the INS server can always serve content to the peers, while in traditional P2P applications, peer has to finish downloading content before sharing with other peers.

8.2.3. Integrated ALTO and INS System for File Distribution

Each INS server can supply the bandwidth usage of at most 94% of network interface card (NIC) - e.g. 1Gbps NIC server can supply

bandwidth of 940Mbps at most. We did tests on 100Mbps and 1Gbps NIC, and got same result of 94% bandwidth usage.

Each INS server can support about 400 online users for file downloading simultaneously. When we tried 450 concurrent online users, 50 users didn't start downloading on time, but wait for the other 400 users to finish downloading.

9. Conclusion and Next Step

This document presents two examples of integrating INS system into P2P applications (i.e. P2P live streaming and Vuze) by developing INS client API for native P2P clients. To better adopt INS system, we found some important design considerations including efficiency for INS connection, control latency caused by INS operations, and developed some mechanisms to address them. We ran some tests to show the results of our integration examples on Amazon EC2 and PlanetLab for deploying INS servers and clients, respectively. It can be observed from our test results that integrating INS system into native P2P applications could achieve performance gain to P2P applications and more network efficient content distribution. For the example of integrating ALTO and INS system to support file distribution of CPs, we have shown the feasibility of such integration.

Our next step work will continue on implementing more features of INS servers and clients based on the protocol ideas developed in protocol document [I-D.alimi-protocol], such as OAUTH based authorization and naming scheme for data objects.

10. Security Considerations

The authorization token can be passed from one INS client to other INS clients to authorize other INS clients to access data objects from its INS storage. Detailed mechanisms of token based authentication and authorization can be found in [I-D.alimi-protocol].

11. IANA Considerations

This document does not have any IANA considerations.

12. References

12.1. Normative References

[I-D.alimi-protocol] Alimi, R., Rahman, A., Kutscher, D., Yang, Y., Song, H., and K. Pentikousis, "DECADE Architecture and Protocol", draft-alimi-protcol-01 (work in progress), June 2013.

[I-D.ietf-alto-protocol] Alimi, R., Penno, R., and Y. Yang, "ALTO Protocol", draft-ietf-alto-protocol-11 (work in progress), March 2012.

12.2. Informative References

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

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

[EC2] "http://aws.amazon.com/ec2/"

[PL] "http://www.planet-lab.org/"

13. References

Authors' Addresses

Ning Zong (editor)
Huawei Technologies

Email: zongning@huawei.com

Xiaohui Chen
Huawei Technologies

Email: risker.chen@huawei.com

Zhigang Huang
Huawei Technologies

Email: andy.huangzhigang@huawei.com

Lijiang Chen
HP Labs

Email: lijiang.chen@hp.com

Hongqiang Liu
Yale University

Email: hongqiang.liu@yale.edu