Uppsala Student Project 2014
NetInf Video Streaming for Falun 2015

Project Specification

Industry representatives:

Börje Ohlman
Ericsson Research

Adeel Mohammad Malik
Ericsson Research

Bengt Ahlgren
SICS Swedish ICT
# Table of Contents

1. Introduction .................................................................................................................. 1

2. Architectural components .............................................................................................. 2
   2.1 Recording/Playing Android application ................................................................. 2
   2.2 Recording/Playing clients ....................................................................................... 2
   2.3 NetInf core network, NRS & NetInf routers ......................................................... 3
   2.4 NetInf DHCP ......................................................................................................... 4
   2.5 Event browser ....................................................................................................... 4
   2.6 NetInf transport .................................................................................................. 5
   2.7 Storage eviction ................................................................................................... 5
   2.8 Video encoding and chunking ................................................................................ 6

3. Code considerations ........................................................................................................ 7
   3.1 Available implementations (from previous projects) ............................................. 7
   3.2 Optimizations ....................................................................................................... 7
   3.3 Revision system ................................................................................................... 8

4. References ...................................................................................................................... 8
1 Introduction

The main goal of the project is to offer a live video streaming Android application that delivers the video over an Information-centric NetInf network [1]. If successful, we hope to deploy this at the Nordic World Ski Championships in Falun in February 2015 [2] where it potentially could be integrated as a part of the Virtual Arena.

Live video from any Android client should be distributed using Erlang NetInf routers to any numbers of simultaneous viewers. Scalability is of key importance here and it can be achieved with Request Aggregation and Caching in the NetInf routers.

There are some existing prototypes that can be taken as a starting point and evolved. We hope to be able to distribute the Android applications to real users via an Android market. The current prototype has very long delays for the streaming video; we will be looking for ways to reduce this delay.

Figure 1: Network architecture
2 Architectural components

This section explains the different architectural components of the system in more detail.

2.1 Recording/Playing Android application

- Android clients for recording and playing the video should be implemented. The recording client will allow the user to record a live stream and publish it while the playing client will allow the user to fetch a stream that has been or is being recorded by another client. The recording and the playing clients can either be integrated within the same Android application or exist as separate Android applications.

- The playing application should provide an Event Browser (see Section 2.5) interface for the users to fetch videos. The Event Browser can either be integrated with the playing client application or exist as a separate application accessible from the playing client application.

- The recording/playing application should use NetInf transport for publishing and fetching data. This can be achieved by using the Java NetInf library. NetInf transport should be running as an always-on background process and not natively in the recording/playing application. This will allow several NetInf applications to use the same instance of NetInf transport.

- The recording/playing clients should use MPEG-DASH (MPEG Dynamic Adaptive Streaming over HTTP) or HLS (HTTP Live Streaming) (see Section 2.8) for segmenting, re-assembling and streaming the video. For this a number of alternatives can be explored such as the Android ExoPlayer [3] and the MP4Box multimedia packager [4]. Note that the HLS/MPEG-DASH layer must be adapted to run over NetInf transport which implies that code modifications and/or extensions will be needed in the reused code.

2.2 Recording/Playing clients

- The recording client should break down the recorded video into chunks. Each chunk will represent a Named Data Object (NDO) with a unique name. These chunks will then be published. The recording client should be able to publish the chunk locally or onto a designated NetInf node in the NetInf core network using “full PUT”. Full PUT refers to the process of including the actual NDO content while publishing an NDO on a remote NetInf node. For each chunk published, a name-location binding should be created in the central Name Resolution Server (NRS). A Name Resolution Server (NRS) is a node which stores name-location bindings for NetInf NDOs and is used as a lookup directory to locate NDOs in the NetInf network.

- A video is a stream of individual chunks published at a NetInf node. Just like each individual chunk, the entire video is also represented by an NDO namely the Header NDO. The Header NDO will contain references to several manifest files where each
manifest file corresponds to a specific representation (bit rate, resolution etc) of the video or audio stream. The manifest files contain a list of video/audio chunk NDOs that have been published for that video. The Header NDO can also have associated metadata which will provide information about the video e.g. title, publisher, timestamp, geo-location etc. Publishing a video means publishing the Header NDO of that video. A name-location binding is created for the Header NDO in the NRS.

- When streaming live, viewers should be able to rewind the video to a previous point in time. This would require the manifest files to keep track of all the chunks that have been published for the video so that the playing client can request previously generated chunks. This, however, presents the problem of large manifest files which must be resolved to make the system scale.

- The name of a video chunk NDO should be determined by hashing the content of that chunk. SHA-256 can be used for creating names for that purpose. However, NDOs with dynamic content such as the manifest files cannot be named using hashing.

- As part of this project several issues must be resolved e.g. video/audio encoding schemes to be used, naming of NDOs with dynamic content (e.g. the manifest files), duration of a video/audio chunk, how should video/audio chunk NDOs be referenced in the manifest files, how many video/audio chunks should the manifest files keep track of in the case of live streaming, how can previously generated chunks be retrieved when the video is rewound during live streaming etc.

2.3 NetInf core network, NRS & NetInf routers

- The NetInf core network is an integral part of the NetInf network. It contains a number of different types of NetInf nodes e.g. NetInf routers, a central Name Resolution Server (NRS), NetInf DHCP. As mentioned earlier, a Name Resolution Server (NRS) is a node which stores name-location bindings for NetInf NDOs and is used as a lookup directory to locate NDOs in the NetInf network.

- The NetInf core network must have a central NRS which stores name-location bindings for NDOs. The central NRS acts as a last-resort NRS and must therefore store bindings for all the video data published in the NetInf network. The central NRS should store bindings for all the published data but may not necessarily store bindings for cached data. This will help the central NRS scale with a large number of users in the system. By storing bindings of Header NDOs, the central NRS will effectively store meta-data associated with all the published videos. This can provide a basis for the Event Browser to function as intended (see Section 2.5).

- The NetInf routers should have associated storage where video data can be published by the recording nodes and also be cached while in transit. The storage should be split into permanent storage (for published data) and temporary storage (for cached data). The distinction between permanent storage and temporary storage is important so that recorded videos can always be retrieved from the origin server as a last resort and at the same time cached data can be evicted to make space for more popular data. Every
A playing client interested in a particular video can start by sending a request for the Header NDO of the video it is interested in. The next-hop NetInf router along the request path will check to see if it has the requested object in its local storage. If not, it will initiate a similar request. If none of the on-path NetInf routers has the object, the request will finally reach the central NRS which will resolve the name of the requested object into a locator.

2.4 NetInf DHCP

- The NetInf DHCP is supposed to facilitate auto-configuration of recording clients, playing clients and possibly other NetInf nodes installed in the stadium and along the ski tracks.

- Auto-configuration will allow the recording and playing clients to determine their next-hop NetInf routers. Auto-configuration can also ensure even distribution of traffic load between the NetInf routers in the NetInf core network by distributing the recording/playing clients evenly between the routers. This entails that the NetInf DHCP keeps track of the number of recording/playing clients assigned to each NetInf router.

- The NetInf DHCP should be placed in the NetInf core network.

2.5 Event browser

- The playing client application should provide an Event Browser interface which will show a map of the arena along with video thumbnails plotted on the map at locations where the respective videos are being recorded. Users can access videos by tapping on the video thumbnails.

- As mentioned earlier, the Event Browser can either be integrated with the playing client application or exist as a separate application which can be opened from the playing client application.

- The Event Browser should by default only show the videos being recorded live at that instance but at the same time should also provide a time slider bar to view videos recorded in the past. The time slider bar can have two time sliders to specify a time range. On moving the time sliders the map should be updated with video thumbnails of videos recorded during the specified time range.
The event browser shows a snapshot of all the videos recorded in a specific time range. This information can be retrieved from the NRS from the meta-data associated with the Header NDOs (e.g. title, publisher, timestamp, geo-location etc). To ensure that a live snapshot is updated with the latest information, the playing clients should periodically poll the NRS to fetch information and subsequently update the Event Browser interface. This mechanism is also useful when old videos are deleted from NetInf caches due to limited storage. A user viewing the Event Browser map interface should not see a video thumbnail for a video that has been deleted from the origin server. An alternative to polling can be to subscribe to the NRS information using a publish-subscribe protocol. The NRS will send out notifications whenever it is updated (an entry for a Header NDO is added or deleted). This entails that the NRS and the Event Browser implement the Publish-Subscribe protocol.

2.6 NetInf transport

NetInf transport should offer two variants, the UDP convergence layer and the HTTP convergence layer. The UDP convergence layer offers a simplistic interaction model with less overhead suited for exchange of control messages. The HTTP convergence layer, on the other hand, offers a more sophisticated interaction model with flow and congestion control mechanisms that are inherent in TCP. The HTTP convergence layer is suited for transmission of high volumes of data.

- The DHCP REQ, DHCP RESP, SUBSCRIBE and NOTIFICATION messages should employ the UDP convergence layer.
- The NetInf GET, GET-RESP, PUBLISH and PUBLISH-RESP messages should employ the HTTP convergence layer. These messages facilitate transmission of high volume video data where flow and congestion control is imperative.

2.7 Storage eviction

- Storage eviction refers to deletion of data from storage when it’s full or close to getting full and there is higher priority data that needs to be stored instead.
• As mentioned earlier, the storage on NetInf nodes should be split into permanent and temporary storage. Permanent storage is supposed to store published data while temporary storage is supposed to store cached data. Storage eviction algorithms need to prioritize between different data elements when evicting stored data. In other words, the least important data should be deleted to make space for more important data. Cached NDOs can be evicted based on a simple algorithm where NDOs with higher number of hits in a recent time window are prioritized over NDOs with lower hits. Published NDOs stored in permanent storage should be evicted using a more sophisticated algorithm where a video should be deleted in its entirety. Deleting one chunk out of the entire video will render the video irrecoverable and also result in stray video chunks in other NetInf nodes in the network. This can have implications related to the Event Browser where, for instance, a video thumbnail displayed corresponds to a video that is not retrievable.

• To delete a video gracefully (in its entirety), the published Header and manifest NDOs along with all the published chunks representing the entire video must be removed across all the permanent storages in the network. This task can be simplified by publishing all the video chunks of a video in the same node. Name-location bindings for deleted chunks must also be removed from the central NRS.

2.8 Video encoding and chunking

![Diagram](image)

**Figure 3**: Video encoding and chunking

• MPEG-DASH [5] or HLS [6] should be used by the recording and playing clients to encode, segment and stream the video. Both the standards feature adaptive streaming
and use HTTP. MPEG-DASH is codec agnostic and HLS typically uses H.264/AVC for video encoding and AAC for audio encoding.

- Figure 3 illustrates the process of encoding, segmenting and NetInfying the video. Recorded video is encoded first and the resulting byte stream is segmented into video chunks. These chunks are NetInfied by a NetInf wrapper which maps them to NDOs with unique names. At the receiving end the NDOs are unwrapped and the resulting chunks are merged and decoded for the player to play the stream.

- The MPEG-DASH/HLS layer must be adapted to run over NetInf transport which implies that code modifications and/or extensions will be needed in the reused code. For this a number of alternatives can be explored such as the Android ExoPlayer [3] and the MP4Box multimedia packager [4].

3 Code considerations

3.1 Available implementations (from previous projects)

This project is a continuation of work done previously. The work done previously delivered an implementation of the NetInf router in Erlang. The current Erlang router implementation supports publishing, caching, local NRS and features like Request Aggregation and Publish-Subscribe. Students are expected to improve the current implementation to achieve better performance and implement new features needed for accomplishing the goals of this project.

Besides the NetInf router implementation, there is also an Android application that implements the recording and playing clients. The NetInf transport functionality is provided by a NetInf library. The application runs NetInf transport natively which should be modified so that NetInf transport runs independently as an always-on background process instead. This will allow several applications to use the same instance of the NetInf stack. The playing client makes use of the media player in the Android core. This should be modified so that the application provides a media player natively which supports MPEG-DASH or HLS.

Lastly, there is an Android implementation for the Event Browser. This implementation should be modified and/or extended to suit the needs of this project.

3.2 Optimizations

The implementation should ensure that the video streaming delay is minimal and at the same time ensure that streaming is lag-free. This requires optimizing parameters like chunk length and the number of chunks buffered by the playing client so that it buffers just the right amount of data for lag-free streaming.

The Android applications and the NetInf routers should be robust to ensure reliable execution. During previous implementation work, a peculiar problem was experienced where the NetInf transport application was thrown out by the Android OS which resulted in malfunctioning of the recording/playing applications. This problem should be resolved to ensure that the NetInf transport process is “always-on”. The Android application should also be user-friendly, easily downloadable and independent of the Android version.
3.3 Revision system

Code modifications and extensions should be managed using Git. Git repositories have been setup for the available implementations. Before starting to work with the Git repositories that have been setup, each student must set his/her name and email in the Git client [7]. This will ease management of commits on the Git repositories.

Following are the URLs, username and password to access the Git repositories:

- NetInf router (Erlang version):
  
  URL: gituser@tnm002.tnm.verkstad.net:/home/gituser/netinf-erlang
  Username: gituser
  Password: uppsalaProj2014-lvs

4 References


