Web Based one-to-many Data Distribution for Retrieving Continuous Stream or Bulk Data using Rate-Controlled Multicast Transfer: HTTPng Protocol (Architecture) Specification: UCL HIPPARCH Draft Deliverable UCL-2

1 Introduction

The Web Architecture is naturally extensible. We have leveraged off this to build a prototype demonstrator, which is described in this (draft) document. The demonstrator uses a varity of protocols between a variety of standard web components to shew off the benefits of moduler, ALF based protocols, and in particular protocols that exhibit adaption, recveiver driven design principles, and flexibility.

Adaption is shown by the layered coder both for coninuous media driven from the LBL/UCL Video tool Vic, and for web page downloading.

Multicast is demonstrated using the rate controled multicast protocol, which implements a TCP like congestion avoidance scheme, but permits heterogeneity of receivers. Seperation of receviers from sender concerns is achived by indirecting access to web servers both at the receiver (client) side, and at the sender)server) side, using standard Web Proxy techniques.

RTP/RMFP techniques are used to buld the video, audio and the reliable multicast streams. These are documented in the paper at the HIPAPRCH workshop, and the Internet Draft in the IRTF RM working group..

Future work may entail a mix of multicast, unicast and cacheing. We hope to include highly hetergenous links (mobile/wireless as well as high bandwidth) i nthe final demonstrator to be specified in UCL Deliverable UCL-3.

2 Outline

The 2 main applications are

1. a java server
2. a java multicast receiver that delivers received info to a browser residing on the same machine using server push

This is connected togerther by some html files with javascript to demonstrate all this.

Upon startup the java server configures itself from the following file:

```
mcastSetup.txt

sender= merci
senderTCPport= 30007
browserTCPport= 30007
fileDirectory= /cs/research/darpa/waffle/common/tmp/servPushImage/fish
epcohPeriod(s)= 2
```

```
              +-------------+
              |  server     |------------->continous multicasts of dynamic
              |             |                    data to preconfigured addresses
              |  MACHINE A  |--------------------------+
              +-------------+^                         |
                   ^:  |       :-------:               |
                   |   |               :               |
  HTTP/TCP request|   |HTTP/TCP reply    :             |
  for pool file    |   |either:            :UDP onehot  |
                   |   |pool file           :mcasts of  |
                   |   |or:                  :pool files|
                   |   |multicast address     :         |
                   |   v                       :        v
           +-------------+                   +--------------+
           |  netscape   |                   |  receiver    |
           |             |------------------>|              |
           | MACHINE B   |<------------------|  MACHINE B   |
           +-------------+  HTTP/TCP         +--------------+
                           server push
```

```
TTL= 1
MulticastPoolLow= 225.0.0.2 MulticastPoolHigh= 225.0.0.5
MulticastPoolPort= 30006
address= 225.0.0.1 UDPport= 30007 filename= /cs/research/darpa/waffle/common/tm
p/servPushImage/camera2/images3.txt sleepTime= 10000
address= 225.0.0.1 UDPport= 30008 filename= /cs/research/darpa/waffle/common/tm
p/servPushImage/mozilla/mozilla.txt sleepTime= 1000
```

Where,

- senderTCPport - is the port that the server uses to handle http requests. The server will either unicast the requested file back over this port or it will return an http redirect message pointing to the requesting browsers multicast receiver.
- browserTCPport - is the port that the java receiver uses to commnicate with the netscape browser on the same machine. Any multicast files received are delivered to the browser over this unicast HTTP/TCP connection.
- fileDirectory - is the directory on MACHINE A where the pool files are stored
- epochPeriod - is the duration over which all pool file requests are stored before deciding how to handle them(either unicast response or multicast response) at the end of the epoch period.
- TTL - is the TTL assigned to multicast packets
- MulticastPoolLow/MulticastPoolHigh - This is the range of addresses used for multicasting pool files
- MulticastPoolPort - port used for multicasting pool files

2.1 Continuous multicast

The remaining entries in mcastSetup.txt define the continuous multicasts: for example address 225.0.0.1, port 30007 is used to multicast a file every 10,000 ms. The sequence of files to be multicasted is obtained from

`/cs/research/darpa/waffle/common/tmp/servPushImage/camera2/images3.txt.`

When the last file in images3.txt has been multicasted the sequence goes back to the first file in images3.txt.

Now the server is configured so that any http/tcp requests for images3.html result in an http redirect response to `http//localhost:TCPBrowserPort/images.html`.

In addition this redirect message contains a user-defined mime-type x-mcast, the contents of which contain teh multicast address 225.0.0.1 and port 30007. The netscape browser needs to be configured for the mime type x-mcast so that it places this information in a temporary file called `/tmp/images.html.txt`. Then once the receiver has obtained the request for images.html from the browser over the local HTTP/TCP connection it knows to look in `/tmp/images.html.txt` for details of the multicast address to join.

2.2 Multicast of pool files

The addresses to be used for multicasting of pool files are defined in mcastSetup.txt. The number, num_address of these addresses is less than the number, num_file of pool files.

At the end of each epochperiod the number of requests for each pool file are examined, and all pool files with 2 or more requests become candidates for multicast delivery. The winners(i.e. the files actually designated for multicast delivery for that particular request epoch) are obtained as follows 1)The num_address pool files with the highest number of requests are assigned multicast delivery 2)When ordering the files, should any 2 files receive an equal number of requests then preference will be given to any file that was also multicast-served in the previous epoch.

When this ordering process is complete any files designated for unicast delivery are sent from the server to the browser in the traditional manner over the existing HTTP/TCP connection.

In the case of the requests designated for multicast delivery teh server sends the requesting browseer an HTTP redirect response containing the URL:

`http://localhost:TCPbrowserport/poolfilename`.

In addition this redirect message contains a user-define `mime-type x-mcast` the contents of which contain teh multicast address and port. The netscape browser needs to be configured for the mime type x-mcast so that it places this information in a temporary file called `/tmp/poolfilename.txt`. Then once the receiver has obtained the request for poolfilename from the browser over the local HTTP/TCP connection it knows to look in `/tmp/poolfilename.txt` for details of the multicast address to join.

Logical Configuration

In the configuration of figure 1, a single server transmit data to a number of simultaneous receivers with different connectivity. In figure 1 we have placed a receiver in the same LAN as the sender, another receiver connected to the server through an Internet path and a third receiver connected by means of a low-bandwidth point-to-point link.

The users of this system request their data using a web browser, that perform http transactions which end up in scheduling the actual data transmission at a given time. The web browser is also used to receive the data via a multicast-capable Proxy server.

The actual data is transmitted using multicast in order to save bandwidth and server processing power, exploiting receivers aggregation.

The difference in the available bandwidth affect receivers in two ways, depending on the data type: if a continuous stream is being transmitted, a lower bandwidth results in a worse

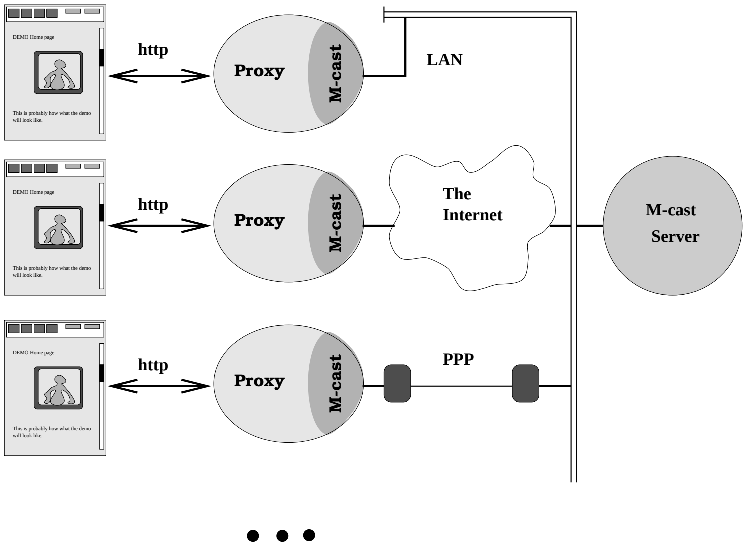

Figure 1:

data quality (we can not have reliable transfer); if bulk data is being transmitted, reliability is possible —often it is a requirement— and lower bandwidth results in a larger transfer time.

Physical Configuration

The logical configuration shown above can be mapped in the following physical setup, see figure 2.

In this configuration all the browser are run on the same machine but they still have a different connectivity because they use different proxies. Proxies are run on distinct machines on which a multicast router is run as well. Each multicast router, receiving multicast data via a separate paths, has the function of feeding the local proxy. The first router is directly connected to the server by means of a LAN and feeds the other routers via multicast tunnels. The Internet path is obtained using a remote host as a reflector. This is obtained by running either a multicast router or an IP tunnel on the remote host.

To set up the configuration of figure 2 we need at least 5 machines, 4 of which should be multicast-capable. All the machine, exuding the one where the browsers are run, should have:

- **OS** *FreeBSD* or *Solaris* (if Solaris they should have Multicast-3.5+ installed, if FreeBSD version 2.0 or later is preferred).
- **Multicast-capable network interface**.

One of the machine above should have two network interfaces.

The machine where the browser is run can have any OS, but it should be quite fast and have good graphical capability.

Moreover we need *Internet connection*, the local hosts should have *valid Internet addresses* and we need the devices to set up at least one point to point connection. They either can be, *analogical modems*, *ISDN modems*, *GSM modems*. If none of them can be provided we can use a *null-modem cable* to implement the point-to-point connection.

Other low-bandwidth connections can also be used, such as an ATM VC.

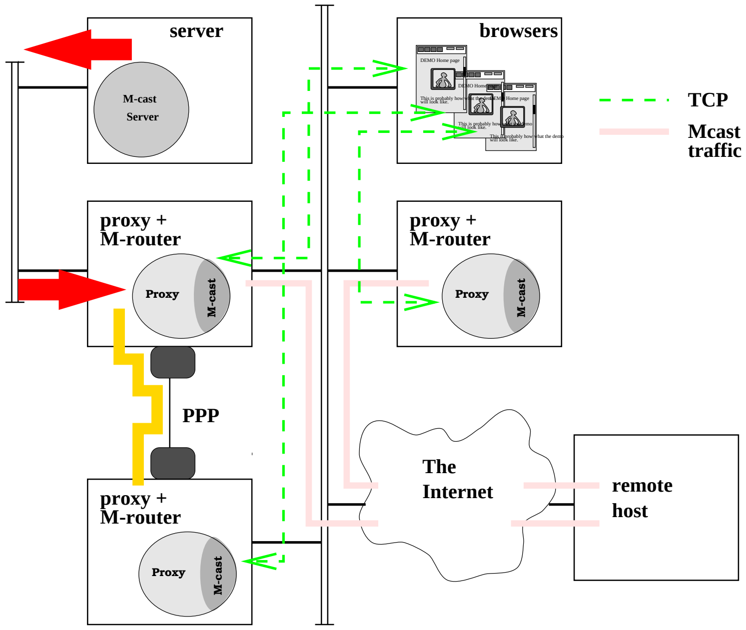

Figure 2:

3 Video Integration

Aim of this part of the demo is to demonstrate congestion control aspects of NoName (NN) multicast distribution protocol [1, 2]. NN performs congestion control by using a layered transmission scheme. Each layer is sent on a different multicast address and consumes different levels of bandwidth. Receivers adapt to network conditions by joining or leaving layers.

3.1 Layering of video

When using a layered scheme to transmit video, the base layer contains low quality video and additional layers provide increasing levels of refinement. This can be achieved either using a layered compression scheme so that additional layers improve the resolution of the decoded image or by using temporal layering of frames. As a layered compression scheme is not currently available, we chose to use temporal layering.

With temporal layering the different transmission layers contain different frames. By joining additional layers, receivers improve the resulting frame rate. This is illustrated in figure 3. The transmitter produces eight frames per second and transmits them on four layers. Joining the base layer (1) results in 1 fps, adding layer 2 improves to 2 fps, 3 improves to 4 and finally adding layer 4 provides the full 8 fps.

To conserve bandwidth vic does not transmit the full image information on each frame. A conditional replenishment algorithm is used that splits the image in blocks and selects for transmission only the ones that have changed significantly. This complicates layered transmission. In order to correctly decode a frame at the receiver the information for the previous frame has to be available. This is not true with the described scheme as if the receiver does not join all layers some frames will be missing.

To work around this problem we have to change the way frame differences are calculated. Figure 4 shows for each frame which the last safe frame to calculate the difference from is. For example for frame 5 we have to calculate the difference from frame 1. In this way even

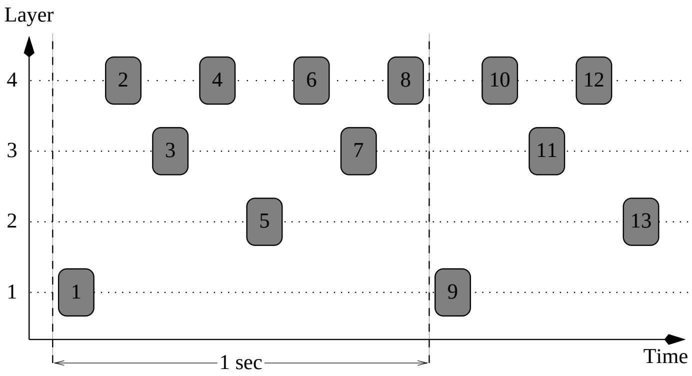

Figure 3: Temporal layering of video frames

if the receiver does not join layers 3 and 4 and misses frames 2, 3 and 4 it can still correctly decode frame 5. The drawback is that redundant information is transmitted as part of the update in frame 5 will be included in 2, 3 and 4.

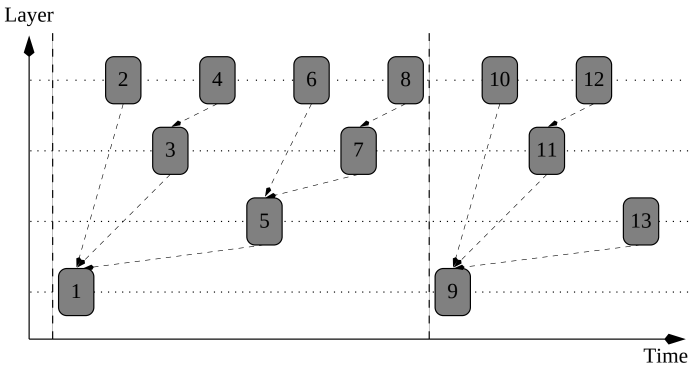

Figure 4: Frame update calculation pattern

In order to be able to calculate frame differences as shown in figure 4 a history of recent frames has to be maintained. To avoid this overhead and simplify implementation in vic we calculate all differences from the last frame transmitted in the base layer. So in our example the difference for frames 2 through 8 is calculated from frame 1.

3.2 Intergration of NN with vic

In order to keep the modifications required to vic small, it was decided not to integrate the NN code in vic but instead to have a separate process perform the protocol operation and communicate with vic through a unix pipe. The setup is shown in figure 5.

References

[1] L. Vicisano and J. Crowcroft. One to many reliable bulk-data transfer in the mbone. In *HIPPARCH '97, Third International Workshop on High Performance Protocol Architectures*, Uppsala, Sweden, June 1997.

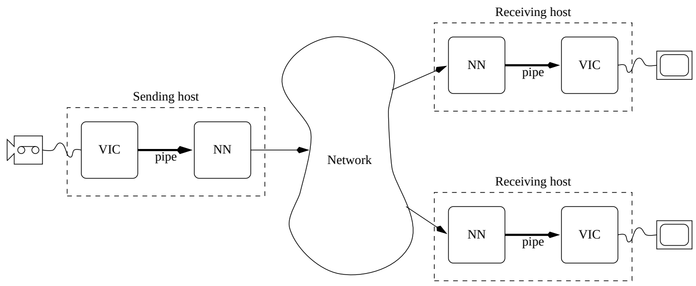

Figure 5: Interaction between vic and NN

[2] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. Research Note RN/97/?, Dept. of Computer Science, University College London, England, August 1997.