UNIVERSITY OF OSLO
Department of Informatics

Design of An Global Multicast Demonstrator for Live Video Streaming on Adobe's Flash Platform

Master Thesis (60 credits)

Liping Huang

April 25, 2013

Design of An Global Multicast Demonstrator for Live Video Streaming on Adobe's Flash Platform

Liping Huang

April 25, 2013

Abstract

Video streaming services has gained more and more popularity in today's Internet. With increasing number of users and video quality, the Internet traffic for video streaming has increased dramatically. At present, most of video streaming services utilize server-client model based on unicast. In a unicast system, the servers suffer from overloading and lack scalability with increased video traffic. Multicast has been considered as an efficient communication mechanism and the basis of a decentralized model. It has been proposed and implemented in video streaming service to overcome the drawbacks in unicast system.

In this work, a multicast demonstrator on Adobe's Flash platform is built to investigate how multicast can improve network efficiency. A series of experiments with the demonstrator are performed in real world Internet. The Internet traffic are are measured and analyzed. The results show that multicast can reduce the load on servers and traffic in network.

Acknowledgements

I would like to thank Trarik Cicic, Carsten Griwodz and Stein Gjessing who have inspired, guided and given support during the course of work. I would like to thank especially Tarik Cicic and his company - Media Network Services AS, providing access to a server on the multicast-enabled network and helps with the experiments.

Finally, I would like to thank my family, especially my husband Gang Lin, my two little princes Frank and Sam,

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In recent years, video streaming is the fastest growing application of today's Internet. The report from Cisco[9] figures out that Internet video traffic account for 51% of global consumer Internet traffic in 2011. There are many prominent examples of video streaming services available in current Internet, such as YouTube[1] and Netflix[2]. In such examples, there are thousands, even millions of users to access video at the same time. The video quality in these video streaming services is also required to near TV resolution standards such as 640×480 pixels for enhanced-definition TV (EDTV) and 1280×720 pixels for high-definition TV (HDTV). As a result, the solutions for video steaming services over Internet must have the ability to support heavy downstream traffic due to many concurrent users and high quality video.

Currently, one of the major solutions for video streaming services is Content Delivery Network (CDN) such as Akamai[3]. It provides a large distributed system of servers across Internet to server user with high availability and high performance. It utilizes the traditional server-client model based on unicast. In such unicast systems, the performance will deteriorate and servers can be easily overloaded due to increasing number of concurrent users and higher video quality.

To solve this problem and increase the scalability, other communication models are proposed and implemented for video streaming such as multicast. Multicast refers to the one-to-many or many-to-many communication. Unlike the traditional server-client model based on unicast, the server in a multicast system does not send all traffic to each client. The multicast system can reduce the need for centralized servers. In the beginning, multicast is purposed to be implemented at the network layer[10] and named as Internet Protocol (IP) multicast. In a IP multicast system, the server sends each packet only once even there are many clients. Each packet is replicated and forwarded on each router, then arrives at a client. IP multicast obviously can handle increasing number of clients without increasing load on servers. However, it increases

[1] http://youtube.com/
[2] http://netflix.com/
[3] http://akamai.com/

the complexities at the network layer and does not support high-level functionalities such as security, access control and network management, etc[11]. In real world, IP multicast can only work in closed networks such as enterprise networks.

Multicast has been proposed to implement at the application layer[7][18][23] and named Application Layer Multicast (ALM) in its early stage. In ALM, each client takes over the tasks of routers in IP multicast, such as replicating and forwarding of packets. Conviva[4], a pioneer to implement ALM as the solution to video streaming, has developed and deployed the live streaming system based on overlay multicast streaming protocols[17][38][40]. ALM provides low latency and in-order delivery of packets. However, the traffic depends on multicast tree consisting of clients. It is very sensitive to node failure in its tree. ALM fits into the definition of Peer-to-Peer (P2P) system, such as delivering packets among all clients across Internet. Some P2P applications such as BitTorrent for file-sharing have gained tremendous popularity and business success. P2P has been proposed and implemented as the solutions for video streaming[24][46][16]. The early ALM based on multicast tree is also known as tree-based P2P multicast. Most of current P2P video streaming systems have utilized a mesh topology and are known as mesh-based P2P multicast.

Compared to IP multicast, P2P multicast can be implemented globally. But it introduces overhead and has higher latency than IP multicast. The combination of IP multicast and P2P multicast can take advantage of the primitive solutions while avoid their drawbacks in the same time, such as Universal Multicast framework[45].

1.1 Goal and Contribution

The goal of this thesis is to implement a prototype demonstrator that provides live streaming services using both IP multicast and P2P multicast. This demonstrator will be also studied and evaluated in Internet. The following contributions are stated in this thesis:

- Literature survey about multicast communication mechanism is presented, with focusing on their key concepts and their opportunities in video streaming.
- Several media streaming systems available in current market are investigated. Some media streaming protocols and P2P streaming technologies are described in detail.
- Prototype demonstrator is implemented on Adobe's Flash platform. This work includes the setup of server and the programming of client applications for live streaming.

[4]http://conviva.com/

- Experiment the prototype in different network environments such as home, academic and commercial network environments in real world. The Internet traffic related to video streaming are measured and analyzed.

1.2 Outline

The rest of this thesis is organized as follows:

- Chapter 2: Relevant literatures about multicast communication mechanism are presented. Several media streaming systems and their streaming technologies are also described, with focusing on Adobe's media streaming systems.
- Chapter 3: The technologies and tools on Adobe's Flash platform is presented. The detailed implementations on server's and client's side are described.
- Chapter 4: A series of experiments using the prototype demonstrator are performed. The analysis and results of their measurements are presented. In the end, some problems met during experiments are mentioned.
- Chapter 5: A conclusion of this thesis and proposals for future work are presented.

Chapter 2

Background

Over the last few years, the popularity of Internet has increased dramatically together with live video streaming. The report from Cisco[9] figures out that Internet video traffic accounts for 51% of global consumer Internet traffic in 2011 and that this percent will be increased to 55 by 2016. The demand for efficient streaming technologies has become very desirable. Multicast is an efficient communication mechanism that meets this demand. This chapter gives relevant backgrounds on multicast and its applications in streaming. In addition, an overview of commercial media streaming systems and their underlying streaming protocols is presented. Motivated by these, this thesis suggests and implements a global multicast demonstrator for live streaming in a commercial media streaming system - Adobe's Flash platform.

2.1 Multicast

Multicast refers to the one-to-many or many-to-many communication. In a multicast system with multiple clients such as n, the server does not sends n identical packets. With one-to-many communication, only one identical packet is sent from the server. Each packet is replicated and forwarded on network's node such as router or client. For many-to-many communication, a client obtains some packets from other clients and these packets are not sent from the server.

Multicast was first proposed by Dr. Stephen Deering in his doctoral work in late 1980's. His work proposed two protocols: IGMP (Internet Group Management Protocol) and DVMRP (Distance Vector Multicast Routing Protocol). These two protocols were multicast solution suite at the network layer[10]. Motivated by Dr. Deering's work, a experimental Multicast Backbone (MBone) was proposed and its first practical application was realized by IETF (Internet Engineering Task Force) in March 1992. The application sent an audio meeting in San Diego to 20 sites worldwide. In the beginning, the MBone was only a virtual multicast network, in which the multicast routing function was provided by workstations. Furthermore, many protocols for IP multicast

have been proposed and deployed, for both intra-domain and inter-domain routings. At present, IP multicast has been included in the standard set of protocols shipped with most commercial routers. The commercial deployment of IP multicast has become possible. However, the deployment in the Internet is very slow due to many practical and political problems.

To solve the deployment problem of IP multicast, it has been proposed to implement multicast at the application layer. In the beginning, the end hosts take over the functionalities of multicast-enabled routers. The end hosts replicate and forward the packets. They also manage the group membership and establish the multicast tree. This multicast is named Application Layer Multicst (ALM). Some examples include Narada[7], NICE[18] and TAG[23], etc. ALM also has its drawbacks such as unfair resource distribution and difficulties in handling frequent churn. ALM has no very successful implementations in real world.

Recently, peer-to-peer (P2P) technology has been adopted for live media streaming with its successful implementation in file sharing and Voice over IP (VoIP). Moreover, both commercial and academical implementations of P2P streaming systems have achieved business success, for example, pplive[29], ppstream[30], coolstreaming[46], etc. Multicast implemented at the application layer is known as P2P multicast. P2P multicast has two categories: tree-based and mesh-based. The tree-based P2P multicast is to establish multicast tree at the application layer, such as the traditional ALM. The mesh-based P2P multicast implements P2P technology similar to BitTorrent in a way.

This section presents multicast at network layer - Internet Protocol Multicast, and its implementations at application layer - Application Layer Multicast and P2P Multicast.

2.1.1 Internet Protocol (IP) Multicast

Internet Protocol (IP) multicast is widely deployed in enterprises, commercial stock exchanges, and multimedia content delivery networks. A common enterprise application of IP multicast is for IPTV such as distance learning and video conferences. IP multicast implements multicast service at IP routing level, i.e.at network layer. With this service, the source only transmits one copy of individual packet. Each packet is replicated at routers and delivered to multiple receivers simultaneously. The key concepts of IP multicast include multicast group address, multicast distribution tree and group membership management.

Multicast Address

Multicast address refers to a logical identifier for a group of end hosts. IPv4 multicast addresses are defined with the leading address bits of 1110 and classified into "class D" addresses, from 224.0.0.0 to 239.255.255.255. Some of these

addresses are reserved for "well-known" purposes by the Internet Assigned Number Authority(IANA). These reserved addresses can be classified into the following ranges:

- The address block from 224.0.0.0 to 224.0.0.255 is locally scoped and the routers will never forward.
- The address block from 224.0.1.0 to 224.0.1.255 is globally scoped and the routers will never forward.
- The address block from 239.0.0.0 to 239.25.25.255 is administratively scoped and used within the confines of an origination. They should never be seen on the Internet.

IGMP - Internet Group Message Protocol

IGMP is used by hosts and their adjacent routers in IP networks to establish multicast group membership. This protocol allows a host and its adjacent multicast-enabled routers to exchange messages. These messages describe the wishes of the host to participate or leave the multicast groups. The network architecture using IGMP is illustrated in Figure 2.1. IGMP is used both at the host side and at the router side. The operating systems at the host side shall support IGMP. Most of currently-used operating systems like FreeBSD, Linux and Windows, support IGMP. The host sends join/leave message to the multicast-enabled router, which is connected to the other part of network. The host joins/leaves the multicast group through the local router.

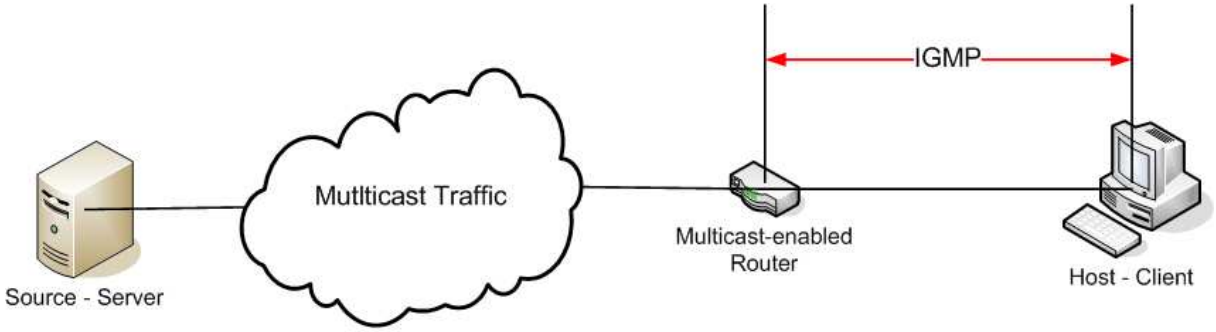


Figure 2.1: Network architecture using IGMP.

This protocol has been developed into three versions, defined by "Request for Comments" (RFC) documents of Internet Engineering Task Force (IETF). All versions include the definition of join/leave procedures for a host. The host joins/leaves the multicast groups by exchanging two defined membership query messages: Membership Query and Report. The join/leave procedures can be described in details as below:

- **join**: A multicast-enabled router sends out periodic queries to all hosts on a given attached network using the 224.0.0.1 multicast address. If one

host is interested in the multicast traffic, it will respond to the query message and return a separate Membership Report to the router. This Report shows that this host is interested in joining the multicast group G from the specified source S. The newly discovered (S, G) will be inserted into the router's (S, G) table. This host joins the multicast group.

- **leave**: When a multicast-enabled router does not hear a report back for a multicast group after some amount of time has passed, the router then clears that (S, G) entry from its (S, G) table. This host leaves the multicast group G.

In the later two versions of IGMP, a few changes have been introduced:

- An explicit "leave" message is introduced to enable the host to leave the group without waiting.
- The qualifying multicast-enabled router with lowest IP address is selected as the querier.
- The security feature ability is introduced and allows each host to select the source address which it wishes to listen to.

Multicast Routing Protocols

Multicast routing protocols are used to construct the multicast distribution trees, which consist of a collection of multicast routers in NM. There are two kinds of multipoint traffic distribution patterns: *Dense Mode* and *Sparse Mode*. The dense mode refers to that many of the nodes are interested in the multicast traffic. The dense mode multicast establishes the source-based distribution tree. The source initially broadcasts to every node. The node not interested in joining the multicast group sends a prune message to the router. After receiving this prune message, the router will not send packets to this node. There are three dense mode multicast protocols: Distance Vector Multicast Routing Protocol (DVMRP), Extends OSPF (MOSPF) and Protocol Independent Multicast - Dense Mode (PIM-DM).

The sparse mode is the opposite of the dense mode and only small percentage of the nodes wish to receive packets from the multicast group. The sparse mode multicast constructs the shared distribution tree based on a core or rendezvous point. The shared tree can be established using existing unicast routing tables. There are two sparse mode multicast protocols: Protocol Independent Multicast - Sparse Mode (PIM-SM) and Core-Based Tree (CBT).

Distance Vector Multicast Routing Protocol(DVRMP)

The DVMRP protocol is the first multicast routing protocol. DVMRP is proposed by Deering[10]. This protocol is described in RFC-1075[43] and has

been implemented in the traditional MBone. It is derived from the Routing Information Protocol(RIP) in [15]. DVMRP is only an experimental multicast routing protocol and uses distance vector technique based on Bellman-Ford algorithm[10].

DVMRP performs the standard *flood-and-prune* procedure to construct a multicast distribution tree, which is called reverse shortest path tree.

- *flood*
  The router will send periodic "Hello" messages on all of its outgoing interfaces using multicast address 224.0.0.4. When receiving this message, the routers perform a reverse path forwarding(RPF) check, which check whether the incoming interfaces on the routers can reach the source in the most efficient path. The interface with successful RPF check will be added into the multicast routing table in the router.

- *prune*
  If some uninterested neighbor routers of one router become interested in joining the multicast traffic, they will send a *graft* message to the router. The router will change the prune status of these neighbor routers in its multicast routing table.

Finally, each router stores a multicast routing table, which gives the shortest path for all multicast groups.

DVMRP is a kind of distance vector routing protocols and suffers from the well-known scaling problems as the other distance vector routing protocols. The reasons are due to that DVMRP needs flooding frequently and has its own flat unicast routing mechanism. In addition, DVMRP uses the *prune* mechanism to determine whether the packets are delivered, and to keep the state information for each source at every router. If a multicast group is densely populated with group members, only a few neighbor routers not in the group need to send the *prune* message. However if a multicast group is not densely populated, most neighbor routers have to send the *prune* message and each router has to store state information in MRT even for non-existing downstream group members. In this case, a significant amount of bandwidth may be wasted.

Protocol Independent Multicasting - Dense Mode(PIM-DM)

PIM is a multicast routing protocol developed by IDRM (Inter-Domain Multicast Routing) working group of IETF. This protocol was proposed in order to operate under any underlying unicast routing protocol and to make use of existing routing/topology tables for RPF checks. PIM supports two different types of multipoint traffic distribution patterns: dense and sparse modes.

PIM-DM is a multicast routing protocol for dense mode and very similar to DVMRP. The two major differences from DVMRP are:

- PIM-DM can use any existing unicast routing table for RPF checks and DVMRP has to keep its own multicast routing table.

- In PIM-DM, each router firstly sends the flood message to all of its own outgoing interfaces. Then the neighbor routers perform RPF checks and generate prune message if RPF checks fail. In DVMRP, each router firstly performs RPF checks for all its neighbors and only sends flood message to the neighbor routers with successful RPF checks.

Protocol Independent Multicasting - Sparse Mode(PIM-SM)

PIM-SM provides a more efficient mechanism for the sparse mode multicast traffic. It defines a "rendezvous" point (RP) between the source and the receiver for each multicast group. One multicast group has only one RP, which is aware of the source. With RP as the core router, a shared tree - Rendezvous Point Tree (RPT) can be established between the RP and the interested receivers. Now if a receiver will join one multicast group, it is unnecessary to know the address of the source. The receiver only needs to notify RP by sending explicit "JOIN" message to the RP. The RP stores the receiver's routing information. The packets are forwarded from the source first to RP with unicast routing, and then to the receivers with the stored routing information via RPT. This process is illustrated in Figure2.2.

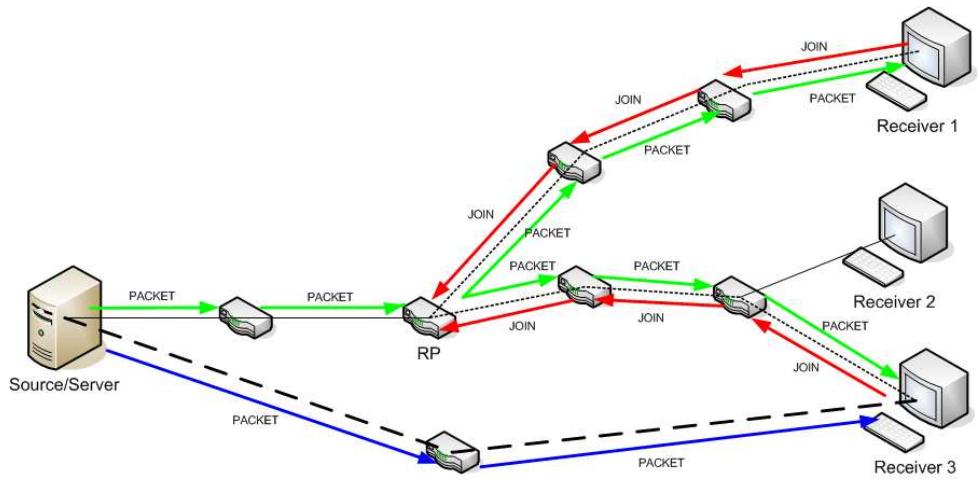


Figure 2.2: PIM-SM process: The interested receivers 1 and 3 send explicit "JOIN" message to the RP via RPT. The packets are sent firstly to the RP and then to the receivers 1 and 3. After a while, the data transmission between the source and the receiver 3 can use the shortest path along the blue arrows.

In addition, PIM-SM provides another option by shifting to the native data transmission via SPT. The RP will quit the data transmission process by sending a Register-Stop message to the source. This mechanism can make sure the more efficient transmission between the source and the receiver and avoid the heavy traffic on the RP.

PIM-SM is a widely used sparse mode protocol and has some advantages compared to other two dense mode protocols. Firstly, sparse mode protocol provides better scalability and only routers on the path between the source and the receiver need to keep membership state. Secondly, sparse mode protocol only forwards packets to the router which has sent explicit "JOIN" message. However, the sparse mode protocol depends on the core router a lot and therefore this core router can be a single point of failure.

Limitations of IP multicast

IP multicast is efficient and can result in superior bandwidth utilization. In the beginning, it attracted significant attention and was proposed as a promising solution for live video streaming. However, its deployment remains limited due to many practical issues. *Diot et al.*[11] figures out an extensive overview of problems in current implementation and deployment of IP multicast. According to [11], current IP multicast lacks simple and scalable mechanisms for supporting access control, security, address allocation and network management. They come to the conclusion that IP multicast does not satisfy the requirements of Internet Service Provider (ISP). IP multicast might be deployed inside an Autonomous System (AS), but not widely adopted outside an AS.

In addition, IP multicast increases the router complexity. Each router has to maintain its associated group members and exchange this information with other routers. This violates of the stateless principle of routers and the best-effort mechanism of IP network.

In conclusion, IP multicast can not be applicable to global streaming services. But IP multicast can be used for streaming services provided by a single ISP.

2.1.2 Application Layer Multicast (ALM)

Application-Layer Multicast(ALM) implements multicast service at the overlay network layer. With this service, the overlay network consists of the hosts which participate in a multicast group. The data transmissions happen among pairs of hosts using unicast from Internet routers. At the same time, these hosts handle other multicast functionalities such as group management, routing and tree construction, without any support from Internet routers. ALM is also commonly known as Overlay Multicast (OM) or End System Multicast (ESM).

Overlay Networks

Recently, great interest has been incurred in applying overlay networks for both the researches and industries. An overlay network refers to a computer network built on top of one or more existing networks. The overlay network

is a virtual network which consists of nodes and virtual or logical links as illustrated in Figure 2.3.

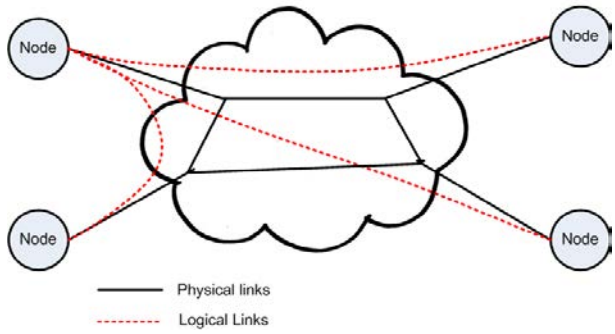


Figure 2.3: An illustration of overlay network: the circles are nodes in a physical network and are linked actually using the physical links inside the cloud, which are presented using solid lines. For an overlay network, all nodes can be seen as linked with the logical links presented by the dotted-lines without thinking about the real physical links inside the cloud.

The current IP network is a historical example of an overlay network, which is built on local area networks, such as Ethernet, phone lines. However, the current overly network usually is referred to as the overlay layer which is built on the native overlay[22]. The native overlay is the IP overlay which provides a best-effort delivery service between the end hosts. There are many proposed overlay network applications, such as multicast, content delivery networks, quality of service, and robust routing etc. The benefits of applying overlay networks include:

- The overlay network can maintain the stateless nature of the native IP network. There is no need for deploying new equipment, or modifying existing software/protocol.

- It becomes possible to introduce the overlay network services only for the nodes which require these services, not for the whole network. New services will not add extra loads to all network equipments, such as memory, bandwidth, etc.

In order to achieve these benefits, some extra costs are needed:

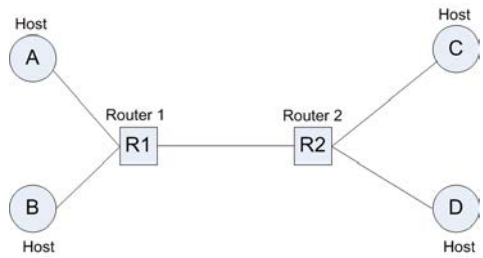
- Adding overheads:
  In overlay architecture, a new layer is introduced between the application layer and network layer. Additional packet headers and control messages have to be used in handling this new layer.

- Adding complexity:
  The overlay layer doesn't eliminate the complexity for the new functionalities and just manages this complexity in this new layer. In addition, there may come unintended interactions between layers.

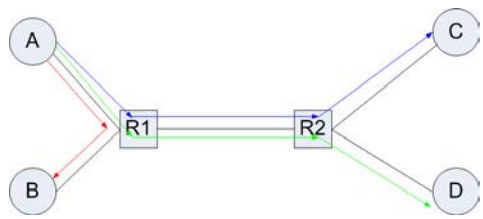
In addition, there have been two main arguments in the network community about the long-term impact of overlay networks in the Internet. One is the purist argument[33], which views the overlay networks as the testbeds for experiments with novel network architectures and as one approach to evolving today's Internet. Another is the pluralist argument[32], which thinks the overlay networks as an internal part of the future Internet and no need for changing today's Internet.

Application Layer Multicast (ALM)

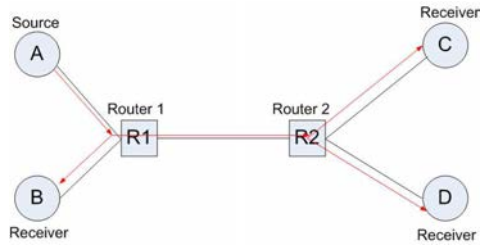
The basic idea of ALM is to replicate data packets at end hosts, instead of routers inside the network as IP multicast does, i.e. moving multicast functionality from the IP network layer to the application layer. The multicast tree is built between participating group members linked using virtual or logical links. Data delivery is using unicast tunneling mechanisms between the participating group members. The comparison of Unicast, IP Multicast and Application Layer Multicast is shown in Figure 2.4 .



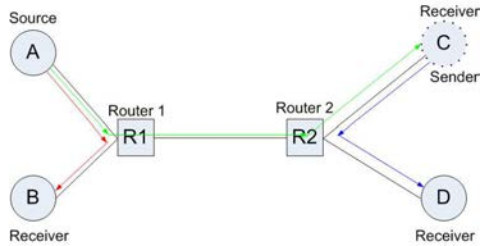
(a) Example: the two squares present the routers and the four circles present the end hosts.



(b) Unicast: three copies of packets are sent from the source A to three other hosts.



(c) IP Multicast: only one copy of packet is sent from the source A, and packets are replicated in the routers and forwarded to other hosts.



(d) Application-Layer Multicast: two copies of packets are sent from the source A, and packets are replicated in the host C and forwarded to the host D. The host C is both a receiver and a sender.

Figure 2.4: Comparison of Unicast, IP Multicast and Application-Layer Multicast.

ALM Techniques

There are three different categories of the current proposed ALM techniques: mesh-first, tree-first and implicit approaches [6]. This section will introduce

three ALM approaches: *Narada*[18], TAG[23] and NICE[7].

Narada

In [18], *Chu et al.* proposed an End System Multicast and a protocol - *Narada* for it, which were one of the first efficient multicast overlays. The End System Multicast is actually a kind of ALMs. *Narada* is designed to achieve all multicast related functionality, i.e. both membership management and packet replication in the Application Layer. This protocol constructs the overlay spanning trees in a two-step process.

1. Construct a *mesh* - a richer connected graph.
   Narada uses the distributed algorithms to construct and maintain the mesh. For each member, a refresh message is generated periodically and exchanged between neighbors. The refresh message contains a list of entr1ies, one entry for every other member in the same group. Use the refresh message to keep the mesh connected, to join the new members, to handle member leave and failure, to repair possible mesh partitions, i.e. group management.
   In addition, a utility function is designed to reflect mesh quality. Members probe each other at random and monitor the utility of existing links or joining new links. The links can be added or removed dynamically to optimize the mesh.

2. Construct the spanning trees of the mesh.
   Use the mesh to construct the optimal trees for the individual source using existing routing algorithms, such as a distance vector protocol. Each tree is rooted at the corresponding source. The per-source trees are constructed from the reverse shortest path between each recipient and the source in the same way as DVMRP. Each member maintains and updates both the routing cost to every other member and the path that leads to such a cost. This can avoid the count-to-infinity problems which are well-known for the distance vector protocol. In addition, the mesh is dynamical and the packet loss could happen when a member leaves or a link is dropped. In order to avoid the packet loss, a new routing cost - *Transient Forward* (TF) is introduced to guarantee that data can continue to be forwarded along old routes for enough time until routing tables converge.

Based on simulations and Internet experiments in [18], it is concluded that the End System Multicast and *Narada* can achieve good performance for small and medium sized groups involving tens to hundreds of members. For larger sized groups the End System Multicast architecture can not keep such good performance. The reason is that *Narada* requires each member to keep information about all other group members. Increased number of group

members results in more memory for storing such information and adds a lot of control overheads.

TAG - Topology-Aware Grouping ALM

In [23] *Kwon et al.* investigated a heuristic application-layer multicast approach, named Topology Aware Grouping (TAG). TAG is a core-based multicast protocol and specifies one root, i.e. primary sender in one group. TAG constructs efficient overlay networks by exploiting underlying network topology information, which means the shortest path information from the given root. These paths are maintained in the routers and determined based on delay as used by current routing protocol. When a new member joins into one group, this member determines its best parent and children based on the paths from the root in that group. There are no needs for extra computations to construct the overlay tree for this new member. Both parent and children are the hosts. The packets are replicated on the parent node and sent to its children nodes. When a member will leave the group, it sends a LEAVE message to its parent. This member is removed from the children list of its parent and its children are inserted to its parent. In this overlay networks, each TAG node maintains only a small amount of state information - IP addresses and paths of its parent and children nodes.

NICE ALM

In 2002, *Banerjee et al.* presented a new application-layer multicast protocol in [7], which was referred to as NICE ALM. This protocol is developed in the context of the NICE project at the University of Maryland, where NICE is a recursive acronym which strands for NICE is the Internet Cooperative Environment.

The NICE protocol is specially designed to support applications with very large receiver sets and low bandwidth real-time data. Such applications include news and sports ticker services, real-time stock quotes and updates and popular Internet Radio sites.

The goal of NICE protocol is to develop an efficient, scalable, and distributed tree-building protocol which does not require any underlying topology information. They create a hierarchically connected control topology by assigning members to different layers from L_0 to L_n. The members refer to the end hosts in the multicast groups. The lowest layer L_0 includes all interested hosts and the highest layer L_n has only a single host. In each layer, hosts are partitioned into a set of clusters. In each cluster, the host with the minimum distance to all other hosts in the cluster is selected as the leader of this cluster. All cluster leaders in the same layer L_i consist of the upper layer L_{i+1}. The clusters and layers can be created using a distributed algorithm described in [7]. The requirements for the distribution of hosts in the different layers are listed as

following:

- A host belongs to only a single cluster at any layer.
- If a host is present in some cluster in one layer, this host must be included in at one cluster in all of its lower layers.
- If a host is not present in one layer, this host cannot be present in any of its higher layers.
- Each cluster has its size bounded between k and $3k - 1$, where k is a constant.
- The number of layers is determined by the size of the cluster and the number of hosts.

Each host in NICE maintains only state about all the clusters it belongs to and about its super-cluster, where super-cluster refers to the next higher layer of this host.

One important property of NICE protocol is to use different overlay structures for control messages and data delivery path based on the given host hierarchy. The given host hierarchy is used as the control topology. In the control topology, each member of a cluster exchanges soft state refreshes with all the remaining members of the cluster. The cluster membership of one host can be created and updated inside the cluster which this host belongs to. The control messages are exchanged among the hosts in the same cluster if the host hierarchy doesn't change. At the same time, a tree can be defined from the given control topology for the given data source. The data delivery can be routed through this source-specific tree.

In addition, the NICE protocol develops also some functions to handle changes of the host hierarchy, such as initial cluster assignment as a new host joins, periodic cluster maintenance and refinement, and recovery from leader failure.

The NICE protocol is another efficient ALM protocol, which can provide some comparable performance with *Narada*. In addition, it has several improvements compared to *Narada*, such as lower control overhead, and potentials to provide the same good performance for large-sized group.

Comparison of IP multicast and ALM

Compared to IP multicast, ALM has the following advantages:

- ALM is one application of overlay networks. It maintains the best-effort mechanism of IP networks. No significant changes for current networks are needed as IP multicast. It's unnecessary for the routers to be capable of multicast and the routing protocol in ALM can use the existing unicast routing protocol.

- All packets are forwarded as unicast packets through Internet and ALM is easy to deploy by implementing multicast functionalities on the end hosts.

- It's possible to support some higher layer features in ALM, such as error, flow and congestion control. These features are difficult to implement using IP multicast.

- The inter-domain multicast in ALM can use the existing inter-domain protocols and IP multicast has to find its particular inter-domain solution.

However ALM has the following disadvantages compared to IP multicast:

- ALM is efficient compared to the unicast, however it cannot achieve the same efficiency as IP multicast does. The control overheads and duplicated packets still have to be delivered over the Internet.

- Potentially increase latency because communication between hosts involves traversing other hosts. The replication of the packets happens on one of hosts. Therefore some hosts have to wait until that host have received them. This introduces more delay for these hosts.

- All hosts have to handle all multicast related functionality, such as group managements, data replications, etc. This introduces increasing complexity on the hosts.

In conclusion, ALM is applicable to global streaming services based on current unicast architecture. It is not as efficient as IP multicast, but more efficient than unicast.

2.1.3 Peer-to-Peer (P2P) Multicast

ALM fits into the definition of P2P systems. A P2P system enables the sharing of data and resources located on all end hosts throughout a network. Such a system allows each end host to contribute resources to other end hosts and requires resources from other end hosts. An end host in a P2P system is named as a peer or a node. All peers in such system are connected at the application layer and forms an overlay topology. The use of P2P technology has gained tremendous success in P2P applications for file-sharing and Voice over IP (VoIP). Recently, the P2P technology has been proposed and deployed into live video streaming. The P2P-based live video streaming is a subset of ALM systems and named as P2P multicast. Existing approaches in P2P multicast can be classified into two categories: tree-based and mesh-based according to their overlay topology[26].

Tree-based P2P multicast

Tree-based P2P multicast closely resembles the design of IP multicast. This approach is to construct a multicast tree on overlay network. The tree is originating from the streaming source. The nodes of the tree are peers. The peer joins the tree at certain level. It receives the packets from its parent peer and forwards the packets to its children peers. The delivery of packets is pushed down the tree from the source. Each peer just forwards every packet it receives to its children peers. It does not schedule packets it receives. It does not query its children peers whether they have received the forwarded packet. This delivery mechanism is referred to as push-based transmission. This mechanism provides low latency and in-order delivery of packets. Tree-based P2P multicast has developed from single-tree based approach to multi-tree based approach.

ESM[18] introduced in Section 2.1.2 is one of single-tree based approaches. Such approach just constructs a single multicast tree at the application layer. One example of tree is illustrated in Figure 2.5. The single-tree based approach suffers two major drawbacks. One is that all leaf peers don't forward the received packets. This implies that the leaf peers don't contribute their upload bandwidth. A large-size P2P system contains a large portion of peers as the leaf peers. This greatly degrades the bandwidth utilization and causes unfair contributions among all peers. The high-level peers have more loads and are more complicated than the leaf peers. Another drawback is that the departure or failure of high-level peers can cause significant program disruption and requires the re-construction of the overlay topology. It has been shown that the recovery is not fast enough to handle frequent peer churn.

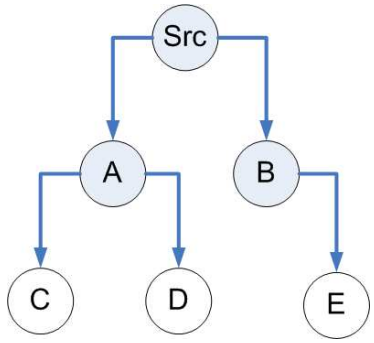


Figure 2.5: Single-tree overlay topology: The circles A, B, C, D and E are five peers. The circle with "Src" represents the streaming source. The packets are sent from the source to all nodes in this tree. The circles C, D, E are leaf peers.

To cope with the drawbacks in single-tree based approach, the multi-tree approach has been proposed[8]. It constructs multiple sub-trees and splits the stream source into multiple sub-streams. Each sub-stream is delivered over a sub-tree. One example with two sub-trees and sub-streams is illustrated in Figure 2.6. Such approach addresses the leaf peers problems in single-tree

based approach. A peer has different position in different sub-tree. It might be a leaf peer in one sub-tree and might not be a leaf peer in another sub-tree. It provides better resilience against peer churn and fairer bandwidth utilization from all peers. However, it introduces the complexity in the other aspects. For example, it requires special multi-rate or/and multilayer encoding algorithms for splitting stream. It also requires disjoint multicast trees, which can be difficult in the presence of network dynamics. Therefore, multi-tree based P2P multicast has not been demonstrated feasible in real system over the Internet.

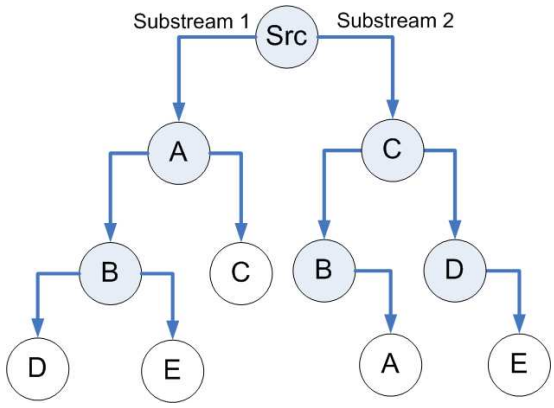


Figure 2.6: Multi-tree overlay topology: The circles A, B, C, D and E are five peers. The circle with "Src" represents the streaming source. The source is divided into two sub-streams and two multicast trees are constructed. The packets from each sub-stream are sent from the source to all peers in each tree. The circles without filling represent leaf peers.

Mesh-based P2P multicast

Mesh-based P2P multicast is motivated from the success of file swarming mechanism, such as BitTorrent. This approach is to form and maintain a mesh topology among peers. One snapshot of a mesh overlay is illustrated in Figure 2.7.

In a mesh overlay, each peer connects to multiple neighboring peers. A peering connection is established based on the mutual agreement between two peers. Once the peering connection is formed, two peers exchange keep-alive messages regularly. If a peer does not receive keep-alive message from another peer within a pre-configured timeout period, this indicates that another peer leaves or fails. The peer will find new neighbors to keep the desired level of connectivity, such as the number of neighbors. The mesh overlay is not static and changes over time.

A peer may receive packets from neighboring peers and forward packets to these peers simultaneously. There are two major ways of data exchange between the peer and its neighbors: push and pull. With the push approach, a peer actively forwards a packet it receives to its neighbors. The peer does not check whether its neighbors have this packet. Therefore, a peer could receive the same packet from two peers. This results in a waste uploading

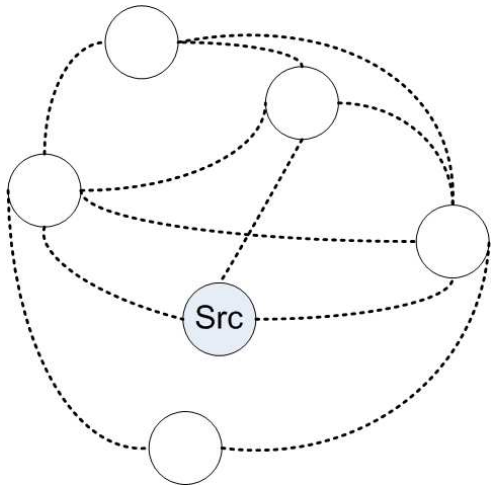


Figure 2.7: A snapshot of a mesh overlay: The circles without filling are five peers. The circle with "Src" represents the streaming source. The dotted lines represent the peering connections. The connections are formed and discarded over time. This mesh is not static and changes over time.

bandwidth in redundant pushes. The pull approach can avoid redundant pushed. With this approach, a peer and its neighbors periodically exchange information about their packet availability using buffer maps. After the peer obtains the buffer maps from its neighbors, it can find which packets it lacks can be found in its neighbors. The peer will send a request to one neighbor and pull a missing packet from the neighbor. However, the pull approach introduces more signaling overhead and additional latency. A peer and its neighbors have to exchange buffer map frequently. The pulling requests are introduced in addition to video data. Moreover, the pulling of a video packet has to wait at least one round delay until the pulling request is responded. It is also possible to combine push and pull data exchange approaches. This approach is named hybrid push and pull approach.

There are many recent P2P streaming systems that adopt mesh-based approaches[27][46][31]. *Magharei et al.* in [28] compare the performance of tree-based and mesh-based approaches using simulations. They come to the conclusion that the mesh-based approach has a superior performance over the tree-based approach. This result is reasonable mainly based on the following points. Firstly, a streaming system with mesh can has better ability to cope with churn. A peer has two or more neighbors. If a peer's neighbor leaves, the peer can still receive packets from remaining neighbors. It does not need to immediately reconstruct the overlay as in multicast trees. Secondly, each peer in such systems has similar functionalities. All peers can make use of their uploading bandwidth and obtain fair bandwidth utilization. Finally, the mesh topology is dynamic. The peering connections are formed and discarded over time. If peers departure or fail, new peering connections will be established among remaining peers. If new peers join, they will be connected to old peers.

2.2 Media Streaming Systems

A media streaming system refers to a set of applications in that a provider constantly delivers media data to the end hosts. At the same time, the media data is played immediately at the end hosts. A typical media streaming is illustrated in Figure 2.8. The media data can be stored files or captured directly from camera. The stream data is encoded for transmission, storage and encryption. The encoded stream is decoded for playback at the end host. The device or software for encoding and decoding media is a codec. The media server is application software that can distribute the media streams to many end hosts. The hosts have client applications that receive and play the media streams. The hosts can be desktops, connected TV, tablets, and smartphones, etc. In addition, an important technology in the media streaming system is the streaming protocol. The protocol mainly define how to deliver the media data over Internet and how to control the deliver in order to obtain smooth playback.

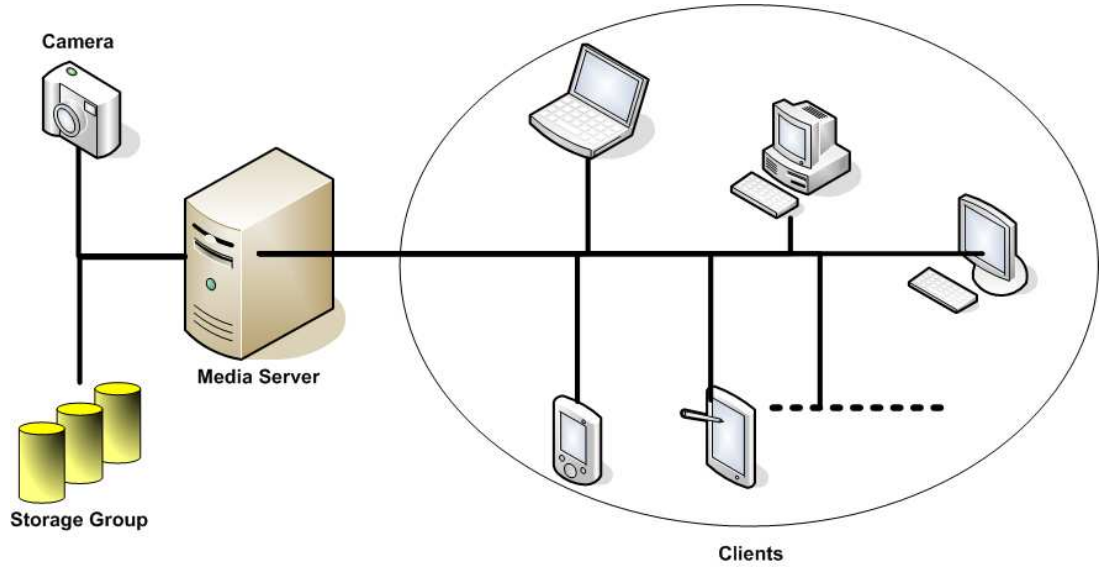


Figure 2.8: Illustration of a typical media streaming system.

2.2.1 Media Servers

A media server is an application software that distributes the media streams. The software usually is located in a computer with server OS. It is also called an application server. The popular media servers in current market include Microsoft's IIS(Internet Information Services) Media Server, Adobe's Flash Media Server, Wowza, etc. They will be introduced briefly in this section.

IIS Media Services[36]

IIS Media Services are media streaming service packages for Microsoft's Windows Servers. They include a web server application and set of feature extension modules. The modules provide an integrated HTTP-based media delivery platform. The packages support Smooth Streaming protocol for streaming over HTTP. They also support Microsoft's Silverlight and Apple's iOS at client sides. In addition, Microsoft provides Software Development Kit (SDK) that allows developers to create their own applications. The newest released version - IIS 7.5 is included in Windows 7 and Windows Server 2008 $R2$.

Flash Media Server (FMS)[1]

Flash Media Server is an application server from Adobe. It supports multiple streaming protocols, including Real-Time Message Protocol (RTMP), Real-Time Media Flow Protocol (RTMFP), Adobe HTTP Dynamic Streaming and Apple HTTP Live Streaming, etc. Its major supported client applications are based on Flash Player. Adobe also provides APIs based on ActionScript Language that allow the developers to create their own applications. The newest version of Flash Media Server is Adobe Media Server 5. It has four editions with different supported features: Adobe Media Server Standard, Adobe Media Server Professional, Adobe Media Server Extended and Adobe Media Server Development Starter (free). This thesis uses the earlier version - Flash Media Server 4.5. It has also four editions with different names: Adobe Flash Media Streaming Server, Adobe Flash Media Interactive Server, Adobe Flash Media Enterprise Server, and Adobe Flash Media Development Server (free). Flash Media Server can work on Microsoft Windows Servers and Linux Servers.

Wowza Media Server[44]

Wowza Media Server covers the most broad range of streaming protocols and clients technologies with lower price. It is a tightly $64-$bit Java server and requires Java Runtime Environment (JRE) or Java Development Kit(JDK). Due to the portability of Java, Wowza Media Server is supported by almost all available server OS. The newest version is Wow Media Server 3.

2.2.2 Streaming Protocols

A streaming protocol defines how to deliver the media data over Internet. It is usually on top of transport layer protocols. The main tasks of streaming protocols are to pack the media data into packets for transmission, to control the delivery of packets for a continuous playback, etc. This section introduces several standard streaming protocols: Real Time Transport Protocol (RTP), Session Initiation Protocol (SIP) and Dynamic Adaptive Streaming over HTTP (DASH). These standard streaming protocols have been implemented in

current commercial streaming systems. The standardization of streaming protocols is important for business success of streaming services. As a pioneer in the field of streaming service, Adobe has designed and implemented steaming protocols for its Flash platform: Real Time Messaging Protocol (RTMP) and Real Time Media Flow Protocol (RTMFP). Two protocols from Adobe will be also described in this section.

Real Time Transport Protocol (RTP)

The RTP is a standardized transport protocol for real time application defined in RFC 3550[35]. It defines a standardized packet structure for delivering the media content. It appends header fields to the audio/video chunks before the sender passing them to the transport layer. These header fields include payload type, sequence number, timestamp, synchronization source identifier and some miscellaneous fields. RTP header is illustrated in Figure 2.9. Such an RTP header along with an audio/video chunk forms an RTP packet. The RTP packet is then encapsulated into a UDP segment.

Payload type	Sequence number	Timestamp	Synchronization source identifier	Miscellaneous fields

Figure 2.9: RTP header fields

The RTP only provides an encapsulation mechanism, but without any mechanism to ensure timely delivery of data or other quality-of-service (QoS) guarantees. In addition, RFC 3550 also specifies RTP Control Protocol (RTCP) as RTP's companion protocol. This protocol defines the RTCP packets. These packets do not encapsulate any media content. These packets are sent periodically and contain the reports that announce statistics about the sender and/or receiver. These statistics include number of packet sent, number of packets lost, and interarrival jitter. These statistics can be used by the application developer for different purposes such as monitoring and modifying the transmission rates, diagnosing the network problems, etc.

The RTP is often used in conjunction with the Internet telephony standards, such as Session Initiation Protocol (SIP) and H.323.

Session Initiation Protocol (SIP)

The SIP[34] is a standard application-layer signaling protocol. It specifies SIP messages for establishing, managing and terminating sessions among participants over Internet. The SIP applications include Internet telephone calls, multimedia distribution, and multimedia conferences, etc. The SIP messages do not include the media data and are only used for creating sessions. The

media data transmission usually uses other transport protocols such as RTP. The messages are ASCII-readable and resemble HTTP messages, such as INVITE message, 200 OK message, ACK message, REGISTER message and BYE message etc. All messages require to be acknowledged. The SIP protocol is independent of the underlying transport layer. It can run on UDP or TCP. A basic procedure to establish sessions between two SIP clients A and B is illustrated Figure 2.10.

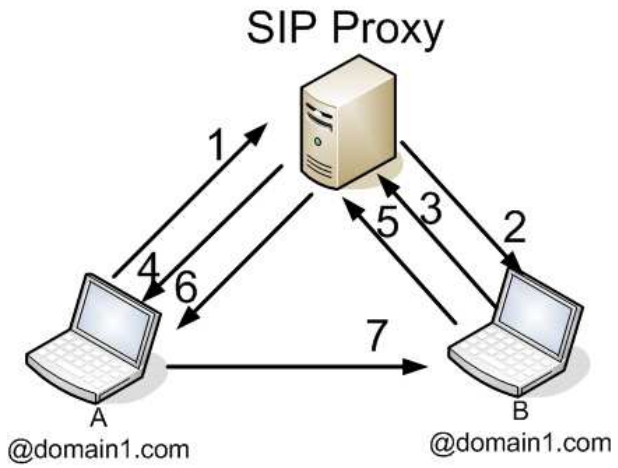


Figure 2.10: SIP basic establishment procedure: the system consists of one SIP proxy and two SIP clients - A and B. The address of SIP clients are e-mail-like address. These steps are taken: (1)A sends INVITE message to proxy. (2)SIP proxy relays INVITE message to B. (3)B sends 180(ringing) message to proxy. (4)Proxy relays 180(ringing) message to A. (5)B sends 200 OK message to proxy. (6)Proxy relays 200 OK message to A. (7)A sends ACK message directly to B. At present, A knows the contact address of B via 200 OK message. A session establishment is confirmed between A and B.

Dynamic Adaptive Streaming over HTTP (DASH)

It has become practical to deliver media content in larger segments using HTTP. HTTP streaming has become a popular approach in commercial streaming services, such as Microsoft's Smooth Streaming, Apple's HTTP Live Streaming[5] and Adobe's HTTP Dynamic Streaming. They all use HTTP streaming as their underlying delivery method. However, they use different manifest and segment formats for different streaming server and clients. MPEG has recently developed a standard for HTTP streaming. This standard is known as MPEG DASH and is expected to be published as ISO/IEC 23009-1[37].

A simple case of DASH is illustrated in Figure 2.11.

The media content is captured and stored on an HTTP server in the form of Media Presentation. A Media Presentation is a structured collection of encoded data of the media content. It consists of a sequence of one or more Periods as shown in Figure 2.12. The media content are divided into the consecutive and non-overlapping Periods according to the time. A Period contains one or more Representations from the same media content. The Representations differ by

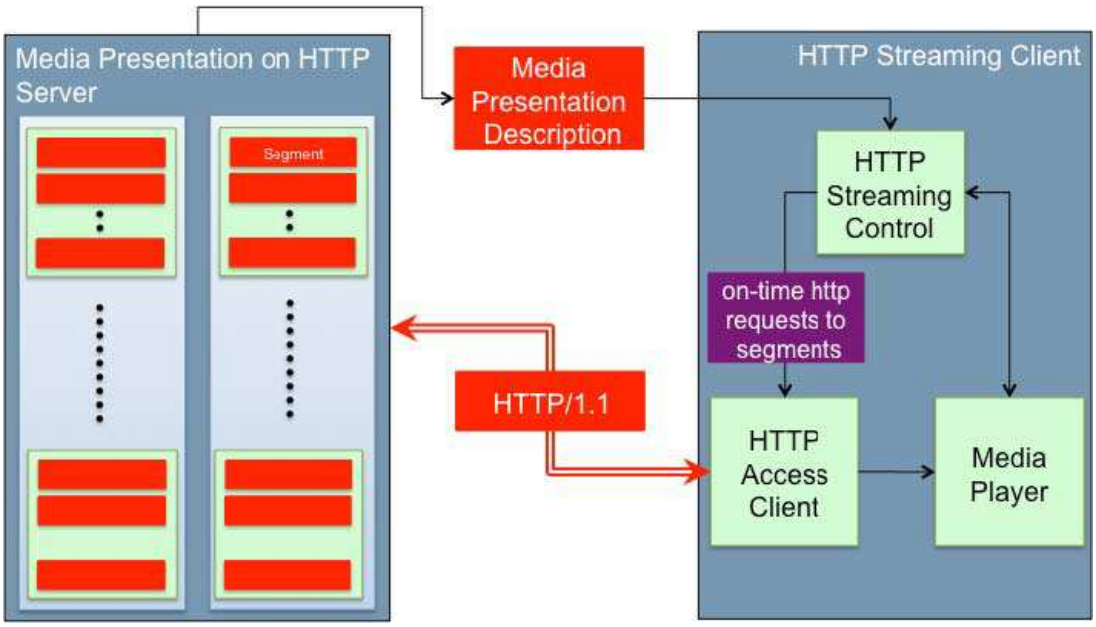


Figure 2.11: A simple case of DASH.[39]

the encoding choice, e.g., by bit-rate, resolution, language, or codec. Each Representation consists of one or more Segments. The Segment is the actual media bit-streams in the form of chunks. It can be uniquely referenced by an HTTP-URL.

In addition to the media data, Media Presentation Description (MPD) is used to describe Media Presentation. The MPD is a manifest which describes the available media content, its various alternatives, their URL addresses and other characteristics. The MPD is usually in the form of XML document. To play the media, HTTP streaming client first obtains the MPD. The MPD can be delivered in many manners, including using HTTP, email, broadcast, etc. The MPD may also be updated during the streaming.

Once the client obtains the MDP, an HTTP streaming control first parses the MDP. The control will select the set of Representations based on the MDP, the client's capabilities, and user's choices. According to its selection, the control sends on-time HTTP requests to the HTTP access client. The access client uses HTTP GET or partial GET methods for downloading media segments. The delivery protocol uses HTTP/1.1 [1]. The media player can play the downloaded segments.

The DASH solution provides an industry-standard for media streaming over Internet with many benefits. Firstly the DASH can easily avoid problems with firewall and NAT traversal, typical in services based on RTP/UDP. It provides the ability to use standard HTTP servers and standard HTTP caches. The most of existing servers and network caches can be reused. Furthermore, it can launch streaming services on the existing successful HTTP-based Content

[1] http://www.w3.org/Protocols/rfc2616/rfc2616.html

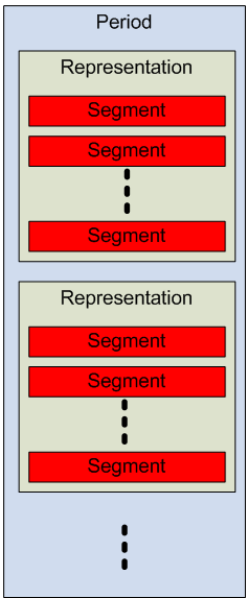


Figure 2.12: A Period of a Media Presentation.

Distribution Networks (CDNs). In addition the DASH moves the complexity to the client. The client can choose the content rate to match its available bandwidth without negotiation with the streaming server.

Real Time Messaging Protocol (RTMP)

The RTMP was initially a proprietary streaming protocol over TCP from Adobe. Adobe has released its incomplete specification in [19]. The RTMP is an application-level protocol for multiplexing and packetizing media transport streams over TCP. It establishes an RTMP connection between the client and server. The connection intends to carry parallel streams of video, audio, and data messages, with associated timing information.

An RTMFP connection begins with a handshake that exchanges three static-size chunks between the client and server. After handshaking, the connection multiplexes one or more chunk streams. Each chunk stream carries messages of one type from one message stream. Each chunk stream is assigned with a unique chunk stream ID. These chunk streams are among a chunk stream for video, a chunk stream for audio, a chunk stream for out-of-band control messages, etc. All chunk streams are delivered independently.

A chunk is basic RTMP packet before transmitted over TCP. A chunk consists of chunk header and chunk payload as illustrated in Figure 2.13. During transmitting, the sender packetizes the messages into chunks (chunking). The maximum chunk size can be dynamically negotiated between server and client. The receiver will assemble received chunks into the original messages. The chunking can split large messages into smaller chunks. This can prevent large low-priority messages from blocking smaller high-priority messages. For

example, it can prevent large video data from blocking small audio data or control messages that have higher priority. The chunking can also reduce overhead of sending small messages. The chunk head contains a compressed representation of information so that such information does not need to be included in chunk payload.

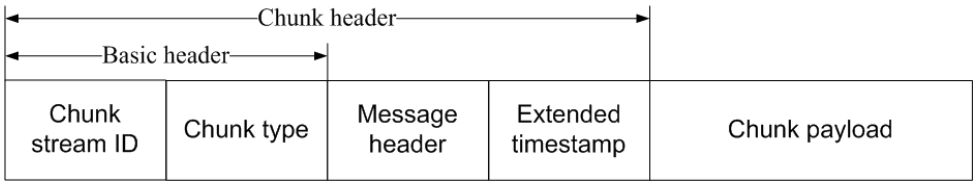


Figure 2.13: An RTMP packet consists of chunk header and chunk payload.

The chunk header contains chunk stream ID, chunk type, message header and extended timestamp. The chunk stream ID is variable-sized. The 2-bit chunk type determines the format of message header. There are four different formats of message header. The message header has variable-size among 0, 3, 7, 11 bytes. It might contain these fields: message stream ID, message type, message length, absolute timestamp and timestamp delta. The message types include audio messages, video messages, command messages, shared object messages, data messages, protocol control messages and user control messages. The extended timestamp field is used when the timestamp in the message header doesn't fit in the 24-bit fields.

In addition, Adobe also introduces multiple variations for RTMP in order to increase its security and compatibility. There are two methods to encrypt RTMP sessions, RTMPS using industry standard SSL mechanisms and RTMPE using Adobe's own security mechanism. RTMPT is encapsulated within HTTP requests to traverse firewalls. It communicates over port 80 and passes the AMF data inside HTTP POST requests and responses.

Real Time Media Flow Protocol (RTMFP)

The RTMFP is a proprietary streaming protocol over UDP on Adobe's Flash platform. It is designed and written based on Secure Media Flow Protocol (MFP) from Amicima. It provides a new transport protocol to securely deliver media flows over Internet. The protocol establishes a session between one pair of peers. The session is a bidirectional channel in which several flows travel as illustrated in Figure 2.14. A flow is a unidirectional communication channel for transporting a correlated series of user messages. The messages can be streams of audio/video, acknowledgements and other messages. This indicates the delivery of the combined signaling and media data via one same channel for RTMFP.

Recently, the author of RTMFP - Michael Thornburgh posts the specification for the RTMFP base transport protocol in [41]. A typical RTMFP packet

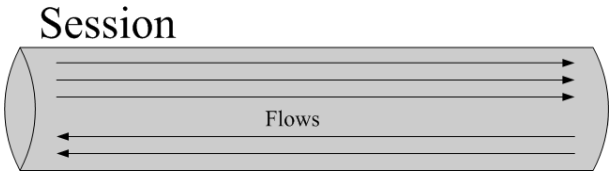


Figure 2.14: Session and flows for RTMFP: many flows are transported inside one session between a pair of endpoints.

is illustrated in Figure 2.15. An RTMFP packet consists of a session ID and an encrypted packet before it is transported over UDP. The scrambled session ID is the session ID modified by performing a bitwise exclusive-or with the bitwise exclusive-or of the first two 32-bit words of the encrypted packet. The session ID identifies the session to which the packet belongs and the decryption key to be used to decrypt the encrypted packet. The scrambled session ID looks more like noise in order to avoid NAT false-positives and annoy Deep Packet Inspection (DPI) boxes. The RTMFP packets are associated with the session ID, not with the IP address. This implies that the RTMFP has IP address mobility. For example, an RTMFP session is up between two peers and the address of one peer has changed. Another peer can find the new address of the peer and continue the data delivery between them.

Figure 2.15: An RTMFP packet consists of a scrambled session ID and a encrypted packet.

The encrypted packet is a packet encrypted according to a Cryptography Profile used in Adobe Flash platform. The packet encryption uses a block cipher operating. The encrypted packets are decipherable without inter-packet dependency, since packets may be lost, duplicated, or reordered in the network. An unencrypted and plain packet consists of a variable-sized header, zero or more chunks and padding as illustrated in Figure 2.16. The header contains flags, timestamp, timestamp echo, etc. A chunk is the data unit of the RTMFP packet that consists of chunk type, chunk length and chunk payload as illustrated in Figure 2.17. The chunk type indicates the content of payload. The chunk can be used for session startup, in-session control and in-session flows. The padding is inserted to meet cipher block size constraints by the sender.

An RTMFP session is established with a 4-way handshake in two round trips as illustrated in Figure 2.18. After the session is established, the initiator and the responder creates new sessions respectively with the initiator and responder session ID. The packets with the corresponding session ID are delivered between them. In addition, the RTMFP also provides other mecha-

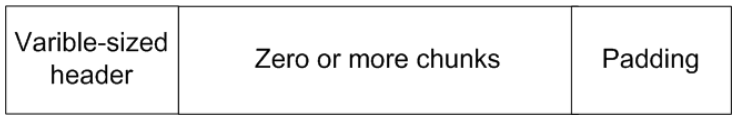


Figure 2.16: A unencrypted and plain packet consists of a variable-sized header, zero or more chunks and padding.

Figure 2.17: A chunk of the RTMFP packet consists of chunk type, chunk length and chunk payload.

nisms for session establishment, such as the forwarder for NAT traversal, the redirector to supply the alternative IP addresses.

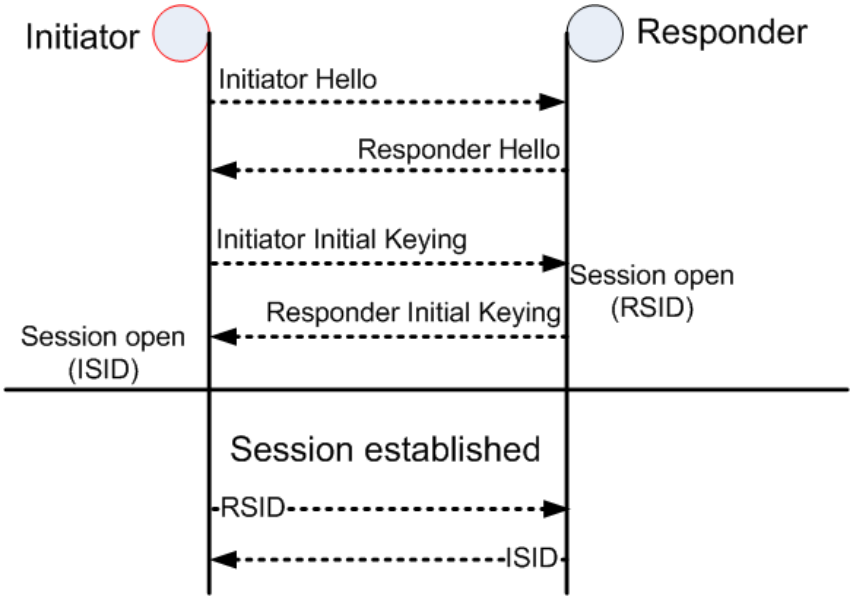


Figure 2.18: A normal session establishment for RTMFP.

Moreover, the RTMFP adds the congestion control and avoidance algorithm according to RFC2914[12] and RFC5681[4]. The congestion control used is TCP compatible. There are two types of acknowledgments for user data of a flow: bitmap acknowledgement and range acknowledgement. The acknowledgements contain the User Data sequence numbers of a flow that have been received. The acknowledgements can notify the packet loss and control the buffer of a receiver.

The RTMFP has integrated the security feature into the protocol itself, including the introduction of the explicit session ID and the encapsulated packet. The use of the session ID instead of IP address supports IP address mobility and prevents hijack.

2.2.3 P2P Streaming Systems

As an efficient communication mechanism, P2P multicast has been applied in the P2P streaming systems in real world. This section presents some representative P2P streaming systems, specially P2P from Adobe. The P2P on the Flash platform motivated the design and implementation of the global multicast demonstrator in this thesis.

Coolstreaming/DONet

Coolstreaming[46] is a data-driven overlay network (DONet) for live media streaming. Each DONet node periodically exchanges information about data availability and partnerships with a set of partners. The node retrieves unavailable data over connections from one or more partners. Both the partnership and the data transmission are bidirectional and flexible.

Each node maintains a membership cache (mCache) that contains a list of active nodes. The nodes in this list do not include all active nodes. The system employs the Scalable Gossip Membership protocol, SCAM[13] to provide a group membership service based on gossiping. Each node periodically generates a membership message to declare its existence. The message contains the node id and number of partners. The nodes which receive message will update the information stored in mCache. In addition, each node also periodically select random nodes from mCache and establish partnership connections with them. This results in a stable number of partners and helps to explore partners of better quality.

Each node also periodically exchanges a Buffer Map (BM) with its partners. The BM represents data availability in the buffer of a node. In the BM, each bit is for a packet. The bit 1 indicates that the packet is available and the bit 0 otherwise. Afterwards, the node retrieves unavailable packet from its partners. The selection of fetching the expected packets from the partners uses a heuristic scheduling algorithm. For each unavailable packet, this algorithm selects the partner with the highest bandwidth and enough available time. The node will fetch the packet from the selected partner.

An experiment system of DONet is implemented in the PlanetLab[2] environment. They evaluate the performance of DONet under stable and dynamic environments. The results show that this system has low control overhead, less than 2% of the total traffic even with over $5-6$ partners. And the control overhead for each node is independent of the overlay size. The playback continuity improves with increasing number of partners. However, the improvement is very limited with more than 4 partners. In addition, the larger overlay will leads to better playback continuity due to more cooperation among nodes.

A public Internet-based DONet implementation was released under the

[2]http://www.planet-lab.org

name *Coolstreaming v.0.9* in 2004. It attracted a remarkable amount of clients, over 1 million. It is implemented using Python. The system can be used under Windows, Linux or other operating systems supporting Python. However, the Coolstreaming service was stopped due to copyright issues in 2005. Now it is the base technology for Roxbeam Inc.[3], which has launched commercial streaming services in multiple countries.

In 2008, a new version of Coolstreaming is presented in [25] with several modifications. The new system discomposes a video stream into several sub-streams by grouping video blocks. Each node in the new system maintains two buffers: synchronization buffer for each corresponding sub-stream and cache buffer that combines sub-streams to one stream. The data transmission in the new system adopts a hybrid push and pull scheme.

P2P on the Flash Platform

Adobe introduces a P2P technology on the Flash platform with RTMFP: RTMFP Group. The RTMFP group defines a mesh overlay network for live media streaming and instant messages. The overlay topology is a ring as illustrated in Figure 2.19. Each node in the ring is represented with a given peer ID. The peer ID is determined by a SHA256 hash of cryptographic public key and other stuff. It is a 256-bit numeric identifier presented as a 64-digit hex strings. This ID cannot be directly chosen or influenced.

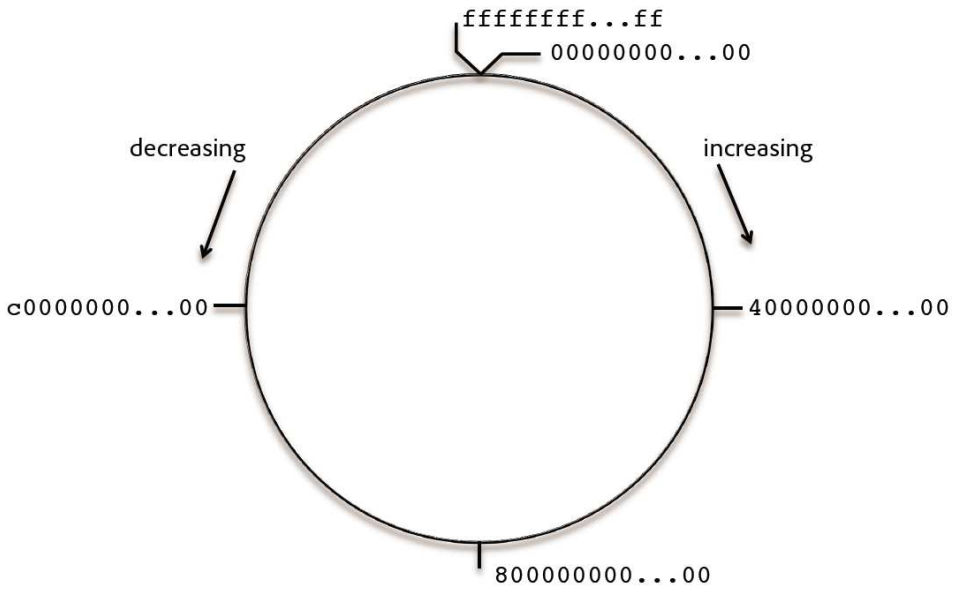


Figure 2.19: RTMFP group overlay topology: a ring mod 2^{256}.(from [42])

Each peer has a list of "heard" peers in the ring. The peer has a topology management algorithm that is repeated forever. The algorithm sorts the

[3]http://www.roxbeam.com

"heard" peers into the "desired" peers and the "undesired" peers. The peer is connected to the "desired" peers. The "desired" peers are continuously optimized through gossip with the rate higher than natural churn. This results in a self-organizing topology. The sorting algorithm chooses the "desired" peers from the "heard" peers with the following rules. The "desired" peers are chosen from 6 closest peers in numeric space, 6 peers with measured lowest RTT and 1 peer that is picked up at random every round. This results in a fully connected ring.

Each peer contains a buffer with the video blocks as illustrated in Figure 2.20. The video blocks are listed in terms of time. The blocks are positioned into two areas: Relay Margin and Window Duration. The peer only attempts to get every block in the Window Duration. If any missed block is outside the Window Duration, the peer no longer attempts to get it. The Relay Margin provides extra time for in-flight requests to be fulfilled.

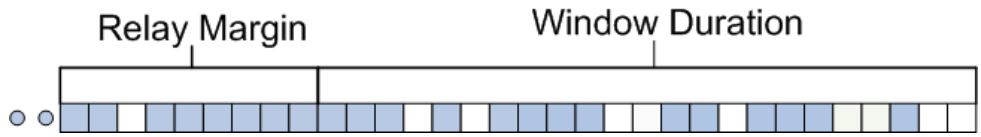


Figure 2.20: Map of video blocks in one peer for P2P on the Flash platform. It is divided into two areas: Relay Margin and Window Duration. Blocks without filling represent current missed ones.

The data delivery between two peers is a hybrid pull and push approach. The data exchanged between two peers are video blocks inside the Window Duration. The pull approach is illustrated in Figure 2.21. In this example, a peer sends a bit map for the video blocks inside the Window Duration to all neighbor peers. A bit with 1 represents the available block in the peer and a bit with 0 for the missed block. A neighbor peer that has received the bit map will pick some of its missed blocks from the original peer and sends the requests for them. The peer responds to these requests and sends the required video blocks to the neighbor peer. The other missed blocks in the neighbor peer might be pulled from its neighbor peers.

The push approach is illustrated in Figure 2.22. For each slice in the Window Duration, the neighbor peer periodically tests its neighbors to check for lower latency. The neighbor peer picks the quickest source peer for each slice and sends a push mask to the chosen source peer. In this example, the neighbor peer chooses the peer as the source for slice number 2nd, 5th and 8th. The neighbor peer sends the push mask to the peer. The push mask on the peer is also updated periodically with lower rate than data delivery. If video blocks for the push mask arrive, the corresponding blocks are pushed to the neighbor peer immediately.

Adobe introduced the P2P multicast since Flash Player 10.1 and Flash Media Server 4.0. In addition, Adobe provided corresponding ActionScript APIs

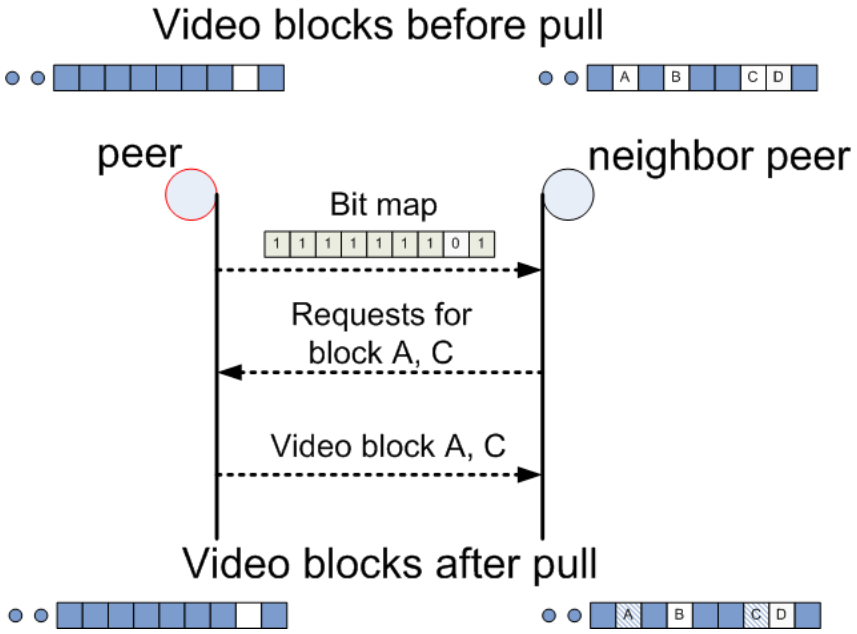


Figure 2.21: The pull approach for P2P on the Flash platform.

that allow the developer to create P2P multicast for live streaming services on the Flash platform.

2.3 Summary

In this chapter, we have presented multicast communication at the network layer and at the application layer in detail. Several media streaming systems and their underlying streaming technologies are described. Among them, we find out that Adobe's media streaming system provides possibility to support both IP multicast and P2P multicast based on its new streaming protocol - Real-Time Media Flow Protocol (RTMFP). Therefore, in next chapter we describe the design and implementation of a prototype demonstrator on Adobe's Flash platform.

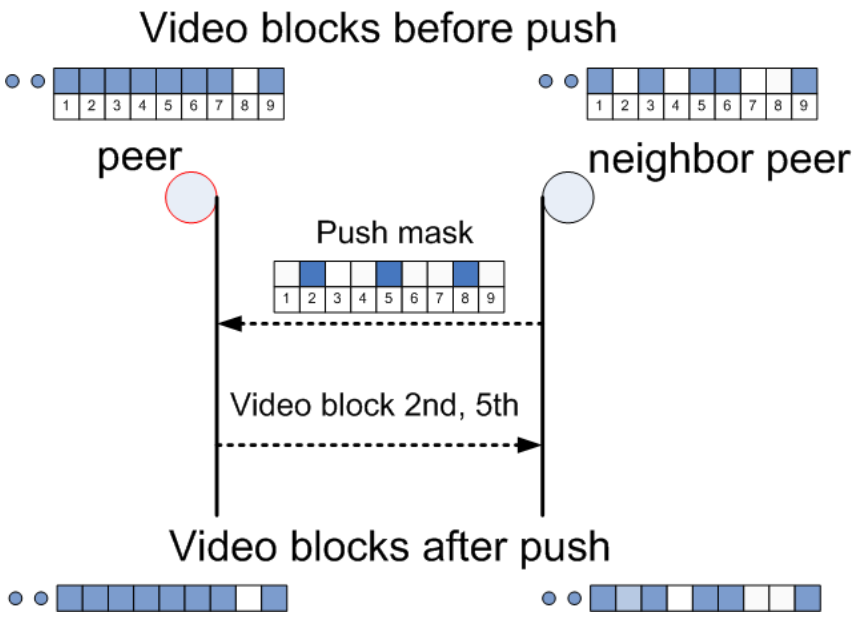


Figure 2.22: The push approach for P2P on the Flash platform.

Chapter 3

Methodology

In last chapter, we figure out the demonstrator is implemented on Adobe's Flash platform. This chapter presents the technologies and tools on Adobe's Flash platform. These technologies and tools that are used for our implementation are described in more detail. Finally, the detailed implementations of the demonstrator's server-side and client-side applications are given.

3.1 Adobe Flash Platform

Adobe Flash Platform[3] is the leading web design and development platform for creating expressive applications, content, and video that run consistently across operating systems and devices and reach over 98% of Internet-connected desktop users. It consists of an integrated set of technologies including client runtimes, tools, frameworks, services, and servers as illustrated in Figure 3.1

In the implementation of the demonstrator, below Adobe's technologies are included:

- Flash Media Development Server 4.5 - a limited, free version for development of Flash Media Server family.
- Flash Player 10.1 and later - the client runtime for the web browser.
- Flash Builder 4.6 - the tool to develop Flash applications running on Flash Player.

This section first presents the above technologies used in the demonstrator. Some important ActionScript APIs for developing the client-side and server-side applications of the demonstrator are introduced.

3.1.1 Flash Media Development Server 4.5

Flash Media Development Server 4.5 contains all the features in Adobe Flash Media Enterprise Server 4.5 software with some limitations. Table 3.1 lists

Figure 3.1: Overview of the Adobe Flash Platform.(from[2])

the limitations related to multicast streaming. It can be used to evaluate and deploy small scale solutions. Its installation file can be downloaded free of charge from Adobe's website[1].

Features	**Limitations**
IP multicast	10 minutes
RTMFP unicast	50 connections
RTMFP P2P introductions	50 connections
Multicast fusion for Flash Player compatible devices	10 minutes
Application-level multicast	10 minutes
Peer-assisted networking	50

Table 3.1: Limitations of Flash Media Development Server 4.5

Installation of FMS

We have installed Flash Media Development Server 4.5 on a computer located at Media Network Service AS. The operating system is Linux CentOS 5.5 64 bit with kernel 1.1. Some hardware information of this computer: 2.40 GHz Intel Xeon processor, 4G RAM and 1-GB Ethernet card. This computer is located on the network with domain name *MNSBONE.NET*. This network supports native IP multicast.

[1] http://www.adobe.com

Our installation uses the default options except the installation of the Apache HTTP server. We use the existing Apache HTTP Server Project[2] on this computer and proxy HTTP connections from FMS to this server. With Apache http, the server can serve media, client SWF files, HTML files, and other page-related files over HTTP. After installation, FMS is started automatically. A Flash Media Server Start Screen is launched on *http://fms.medianetworkservices.com*. This page can be used to check whether FMS is installed and started successfully. Flash Media Administration Server is also installed and connected to FMS. This server listens on port 1111 by default. It also supports the Administrator APIs that can be used to create tools for monitoring and administering FMS.

Administration Console

The Administration Console is an administrative application that calls Administration APIs to inspect and manage the server. It is wrapped into an HTML page and can be launched on *http://fms.medianetworkservices.com/fms_adminConsole.html*. After the administrator logs in, he can manage FMS and view information about applications running on FMS. Figure 3.2 illustrates one snapshot of the console after login. In this snapshot, the console displays the live log of a running application *vod*. It can also present the clients, shared objects, streams and performance of this application if other tab is selected.

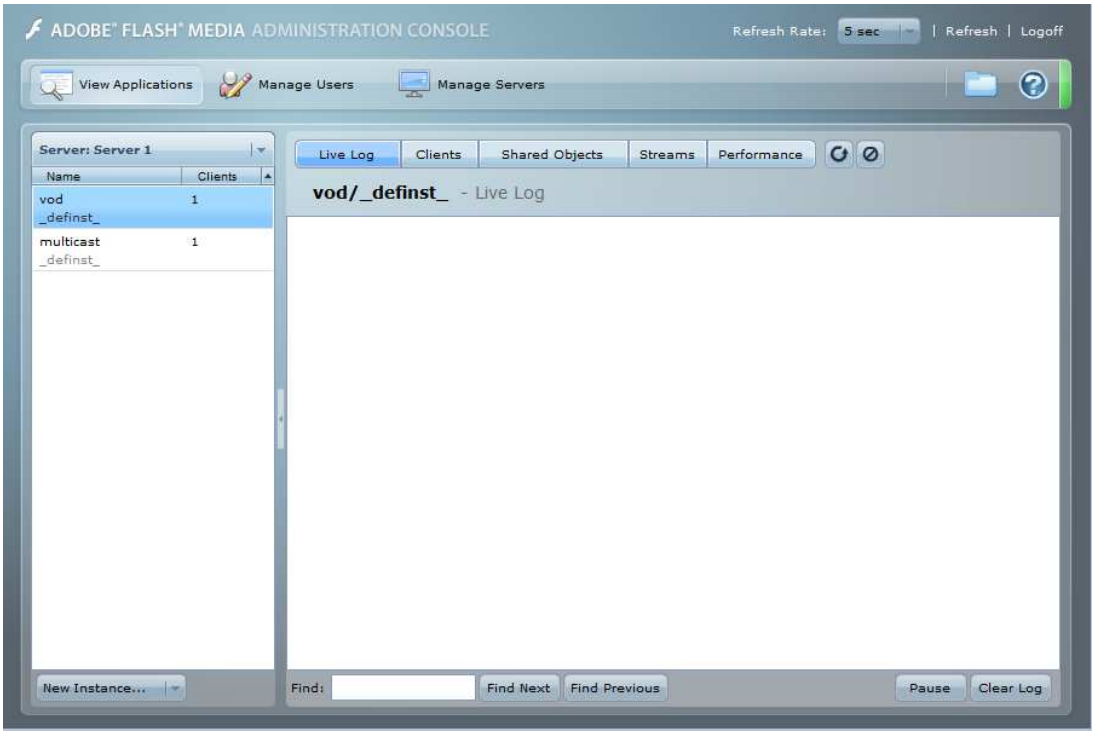


Figure 3.2: Adobe Flash Media Administration Console.

[2]http://httpd.apache.org

Configuration of FMS

FMS can be configured by the configuration files located in the *rootinstall/conf* folder. The folder structure is illustrated in Figure 3.3.

```
[fms@adobe-1 fms]$ tree conf/
conf/
|-- Logger.xml
|-- Server.xml
|-- Users.xml
|-- _defaultRoot_
|   |-- Adaptor.xml
|   `-- _defaultVHost_
|       |-- Application.xml
|       `-- Vhost.xml
`-- fms.ini

2 directories, 7 files
```

Figure 3.3: Default structure of the configuration folder for FMS.

This folder contains two directories and seven files that are described in Table 3.2. These configuration files are in the form of XML. FMS needs to be restarted if any configuration file has been changed.

FMS has a hierarchical structure with four levels: *server*, *adaptor*, *virtual host* and *application*. The *server* is at the top level and contains one or more adaptors. The *adaptor* is assigned an IP address or a port number. The *adaptor* has its own directory inside the *conf* directory. The adaptor directory shall contain an *Adaptor.xml* and a *_defaultVHost_* directory. Each adaptor shall have a default virtual host and other custom virtual hosts. The *virtual host* can not be assigned an IP address or a port number, but must be mapped to a DNS entry or other name resolutions. The virtual hosts are used to host multiple websites on a server, for example, *www.example.com* and *www.test.com* at the same server. FMS uses adaptors to organize virtual hosts by IP address or port number. The virtual host has its own directory inside the adaptor directory. The virtual host directory contains *Vhost.xml* and *Application.xml*. A virtual host can host multiple applications. The application directory for the virtual host can be defined in *fms.ini* and in its Vhost.xml. For example, *fms.ini* defines this directory with *VHOST.APPDIR = rootinsall/applications* and *Vhost.xml* in the virtual host directory with *<AppDIR>${VHOST.APPDIR}</AppDIR>*. This directory is registered directory for the server-side applications.

File/Directory	Description
Logger.xml	a configuration file that contains the elements and information used to configure log files. The information includes the location of the log files, the events written in the log fiels, etc.
Server.xml	a configuration file which affects the entire server unless that is overridden in another configuration files.
Users.xml	a configuration file which identifies administrators and set their access permissions.
*_defaultRoot_*	a directory that contains configuration files for the default adaptor and configuration folders for the virtual hosts inside it. The name of directory is the name of the adaptor.
Adaptor.xml	a configuration file for the adaptor. It determines the number of threads that can be used by the adaptor, the communications ports the adaptor binds to, and the IP addresses or domains from which the adaptor can accept connections, etc.
*_defaultVHost_*	a directory that contains configuration files for default virtual host. This virtual host is mapped to *fms.medianetworkservices.com*.
Application.xml	a configuration file that defines the default settings for all applications within the virtual host.
Vhost.xml	a configuration file that defines the settings for the virtual host. These settings include aliases for the virtual host, the location of the virtual host's application directory, limits on the resources the virtual host can use, and other parameters.
fms.ini	a default configuration file which contains the most commonly edited configuration parameters.

Table 3.2: Configuration folder for FMS

Server-side application

The server-side applications are located in the application directory for the virtual host, for example, the *rootinstall/applications* in this thesis. One server-side application is defined as a directory under this registration directory. The name of the directory is the name of server-side application. This directory usually contains a server-side ActionScript file and other files for configuration. Figure 3.1.1 illustrates the structures of *live* and *multicast* applications. These two applications are sample server-side applications that are installed by default. The server-side ActionScript file is usually a *main.asc* as for *multicast* application and *main.far* for *live* application. The *main.far* is the packaged file using *far.exe* tool. In addition, an *Application.xml* can be located in a particular application directory as examples in Figure 3.1.1. This applications-specific file can override the settings in *Application.xml* in the virtual host con-

figuration directory. It allows changing the settings for a particular application without restarting the server. In addition to two files mentioned above, the application directory can also include other files if needed. These files can be *.txt*, *.xml* or other formats. These files generally contain information that can be accessed by the ActionScript file.

```
[fms@adobe-1 fms]$ tree applications/multicast/
applications/multicast/
|-- Application.xml
`-- main.asc

0 directories, 2 files
```

(a) multicast application

```
[fms@adobe-1 fms]$ tree applications/live
applications/live
|-- Application.xml
|-- allowedHTMLdomains.txt
|-- allowedSWFdomains.txt
|-- main.far
`-- readme.txt

0 directories, 5 files
```

(b) live application

Figure 3.4: Server-side application directory.

The ActionScript file is the heart of server-side application. When a server-side application is loaded on FMS, the ActionScript file is first loaded. The script is written using Server-Side ActionScript based on ECMAScript edition 3 language specification. FMS contains an embedded JavaScript engine - Mozilla SpiderMonkey that can compile and execute this script. When a client connects to a server-side application on FMS, the script of the application is compiled and a new application instance is created. The function *trace(expression)* is used to debug the script. Its expression appears in the Live Log panel of the Administration Console as shown in Figure 3.2.

3.1.2 Flash Player

The Flash Player can play Flash applications in the form of SWF files and is deployed as plug-in in web browser. The SWF files are completed, complied and published files. These files can not be edited any more. The SWF files in this thesis are built and published using Flash Builder and are wrapped into an HTML page. An example of script block to wrap a *SWFname.swf* is shown in Listing 3.1. In this way, the client can launch the SWF files in the HTML pages delivered by Apache HTTP server.

Listing 3.1: An example of script from an HTML file that wraps an SWF file

```
<script type="text/javascript">
  // For version detection, set to min. required Flash Player version,
  //or 0 (or 0.0.0), for no version detection.
  var swfVersionStr = "11.1.0";
  // To use express install, set to playerProductInstall.swf,
  //otherwise the empty string.
  var xiSwfUrlStr = "playerProductInstall.swf";
  var flashvars = {};
  var params = {};
  params.quality = "high";
```

```
  params.bgcolor = "#ffffff";
  params.allowscriptaccess = "sameDomain";
  params.allowfullscreen = "true";
  var attributes = {};
  attributes.id = "SWFname";
  attributes.name = "SWFname";
  attributes.align = "middle";
  swfobject.embedSWF(
    "SWFname.swf", "flashContent",
    "100%", "1302",
    swfVersionStr, xiSwfUrlStr,
    flashvars, params, attributes);
  // JavaScript enabled so display the flashContent div
  // in case it is not replaced with a swf object.
  swfobject.createCSS("#flashContent", "display:block;text-align:left;");
</script>
```

Flash Player Settings Manager

Flash Player Settings Manager provides options to control applications run on Flash Player. It provides several panels to manage global settings, such as privacy settings, storage settings, security settings, and automatic notification settings, etc. Setting Manager is available locally or online. The Local Setting Manager is accessed on Windows, Mac, and Linux computers with Flash Player version 10.3 and later. For example, it can be accessed in the Control Panel on Windows and in System Preferences on Mac. For other OS and earlier versions of Flash Player, the Online Settings Manager is accessed from Flash Player Helper page[3].

3.1.3 Flash Builder

Adobe Flash Builder is an integrated development environment (IDE) for building cross-platform, rich Internet applications (RIAs) for desktops and a wide variety of mobile devices. Flash Builder is built on top of Eclipse and provides all the tools required to develop applications using open-source Flex framework and ActionScript 3.0. It can build many kinds of applications that use Flex framework, MXML, Adobe Flash Player, Adobe AIR, ActionScript 3.0, and LiveCycle Data Services. The client-side applications in this thesis are compiled MXML applications using ActionScript 3.0 and MXML.

ActionScript 3.0[14] is an object-oriented programming language ideally for building rich Internet applications. It is based on ECMAScript which is the international standardized programming language for scripting. MXML is an XML-based user interface markup language introduced by Macromedia. An example source code of MXML application with name *test.mxml* is shown in

[3]http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager.html

Listing 3.2. This application contains a button and a label. To click the button can change the text of the label.

Listing 3.2: An example source code of MXML application

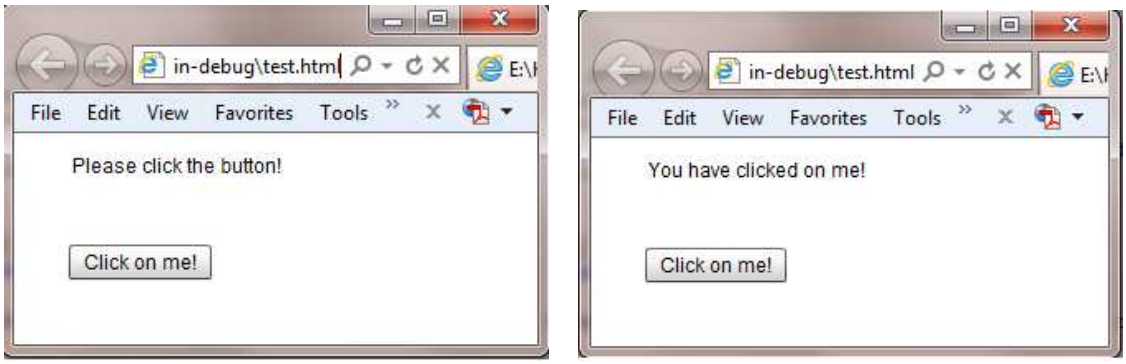
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
          xmlns:s="library://ns.adobe.com/flex/spark"
          xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955"
          minHeight="600" creationComplete="init()">

  <fx:Script>
    <![CDATA[
      private function init():void
      {
        buttonClick.addEventListener(MouseEvent.CLICK, clickHandler);
      }
      private function clickHandler(e:MouseEvent)
      {
        labelStatus.text = "You have clicked on me!";
      }
    ]]>
  </fx:Script>

  <fx:Declarations>
  <!-- Place non-visual elements (e.g., services, value objects) here -->
  </fx:Declarations>

  <s:Button id="buttonClick" x="33" y="67" label="Click on me!"/>
  <s:Label id="labelStatus" x="33" y="10" width="181" height="49"
    text="Please click the button!"/>
</s:Application>
```

Building this MXML file generates a *test.swf* file and a *test.html* file. The SWF file is wrapped into the HTML file in the same way as shown in Listing 3.1. Open the HTML file in IE browser as shown in Figure 3.1.3.



(a) Before click the button (b) After click the button

Figure 3.5: Open test.html in IE

3.1.4 ActionScript 3.0 APIs

This part introduces some classes used in demonstrator's client-side applications referring to ActionScript 3.0 Reference[20].

Camera class

The *Camera* class is used to capture video from the client system or device camera. The captured video can be displayed locally or transmitted. The video quality can be specified by the properties of *Camera*. The following properties are considered in the demonstrator.

- *height*: the capture height in pixels.
- *width*: the capture width in pixels.
- *fps*: the maximum rate at which the camera can capture data, in frames per second.
- *quality*: the required level of picture quality, as determined by the amount of compression being applied to each video frame.
- *bandwidth*: the maximum amount of bandwidth that current outgoing video feed can use, in bytes per second.

Video class

The *Video* class is used to display live or recorded video. The video can be the recorded video on a server or locally, or the live video captured from camera.

NetConnection Class

The *NetConnection* class creates a two-way connection between a client and an application instance on the server, or a two-way network endpoint for RTMFP P2P group communication. It can establish the connection, but can not send streams of media and data. Another class *NetStream* is used to deliver streams over the connection.

NetStream Class

The *NetStream* class is used to open a one-way streaming channel over established connection. A streaming channel is unidirectional and can carry more than one type of content. The content can be audio, video and message data.

NetGroup Class

The *NetGroup* class is used to represent the membership in an RTMFP group. This class can do the following:

- Monitor Quality of Service (QoS) statistics about P2P data transport of this group.
- Broadcast ActionScript message to all members of this group.
- Send a short data message to a specific member of this group. The source and the destination do not need to have a direct connection.
- Break up large data into pieces and replicate it to all nodes in this group.

GroupSpecifier Class

The *GroupSpecifier* class is used to define the parameters and capabilities of an RTMFP P2P group. The important parameters and capabilities for a multicast-enabled group include:

- group name;
- whether multicast streaming is enabled;
- whether peer-to-peer connections are disabled;
- whether members can open a channel to the server;
- IP multicast address if IP multicast is enabled. This address causes the member to join the specified IP multicast group and listen to the specified UDP port.

Once such a object is created, an opaque *groupspec* string can be created using the method *GroupSpecifier.groupspecWithAuthorizations()*. This string starts with "*G* :" followed by hexadecimal digits, for example "*G* : 01010*b*...". If any parameter or capability of *GroupSpecifier* is changed, the new string is created and represents a new group.

MulticastStreamInfo Class

A NetStream object with underlying RTMFP peer-to-peer and IP multicast stream transport is named as a multicast stream. The *MulticastStreamInfo* class presents various Quality of Service (QoS) statistics of a multicast stream on a local node with the following properties:

- the number of media bytes sent or received,
- the number of media fragments sent or received,

- the number of control bytes sent or received,
- the snapshot of the current rate averaged over a few seconds.

NetStatusEvent Class

The *NetStatusEvent* class is a subclass of the *Event* class. A *NetStatusEvent* object is dispatched when a *NetConnection*, *NetStream*, or *SharedObject* object reports its status or its error condition. The status or error is represented using a string contained in *NetStatusEvent* object. For example, when attempting to establish a connection by calling *NetConnection.connect()*, a *NetStatusEvent* object is returned. If the string contained is "*NetConnection.Connect.Success*", it means that the connection attempt has succeeded. If the string is "*NetConnection.Connect.Failed*", the connection attempt has failed.

3.1.5 Server-Side ActionScript APIs

This part introduces some classes used in server-side script *multicast/main.asc* referring to Adobe's Server-Side ActionScript Language Reference[21].

Application Class

The *Application* class is used to represent an instance of a Flash Media Server application. It is created automatically when an application instantiated by the server. It can be used to accept and reject client connection attempts, to register and unregister classes and proxies, and to manage the life cycle of an application.

Client Class

The *Client* class is used to handle each user which is connected to a Flash Media Server application instance. It is created automatically when a user connects to an application on the server. It is destroyed when the user disconnects from the application. An application instance is allowed to have thousands of active clients connected. This class allows the server to determine the properties of each client, such as its version, its platform, and its IP address, etc. It also provides the methods to set bandwidth limits and to call methods in client-side scripts.

NetConnection Class

The server-side *NetConnection* class is used to create a two-way connection between a Flash Media Server application instance and an application server, another Flash Media Server, or another Flash Media Server application instance on the same server.

NetStream Class

The server-side *NetStream* class is used to open a one-way channel for publishing a stream to a remote Flash Media Interactive Server, an multicast group and an RTMFP group. The multicast group is specified using an multicast address and a port.

NetGroup and GroupSpecifier Classes

The two classes for server-side application are the same as these for client-side application. The *GroupSpecifier* class is used to describe an RTMFP group. The *NetGroup* class is used to represent membership of the RTMFP group.

3.2 Environments for Demonstrator

This thesis implements a demonstrator which provides real-time video service on the Flash platform. This demonstrator uses Adobe's new protocol - Real Time Media Flow Protocol (RTMFP) for streaming. The demonstrator also utilizes Adobe's P2P thechnology - RTMFP group. Therefore, the environments around demonstrator must support RTMFP and RTMFP group. This section presents some important settings for supporting these two technologies.

3.2.1 Configure Flash Media Server for RTMFP

Flash Media Server

Port configuration

To establish an RTMFP connection between client and Flash Media Server, the following ports must be available on the server running Flash Media Server.

- *UDP port 1935*: Flash Media Server listens to RTMFP request over this port. This port can be specified in the **ADAPTOR.HOSTPORT** parameter in the *rootinstall/conf/fms.ini* file.

- *UDP ports 19350-65535*: These ports are called RTMFP redirect ports or RTMFP migration ports. Flash Media Server binds one port in this range to RTMFP listener and sends the port number to RTFMP client. Afterwards the client connects to Flash Media Server over this port. This port range can be specified in the *rootinstall/conf/_defaultRoot_/Adaptor.xml* file.

3.2.2 RTMFP Connectivity

NAT (network address translation) and firewall filtering can block RTMFP connections. To ensure a client on a particular network can create an RTMFP connection, the network must satisfy the following requirements:

- Allow UDP traffic. RTMFP is based on UDP.
- NAT or firewall used has predictable behavior. It complies with the NAT implementation recommendations of IETF BEHAVE working group[4].

In addition, Matthew Kaufman, the inventor of RTMFP provides a website called *RTMFP Connectivity Checker*[5]. A client opens this website in a web browser. It checks whether this client can establish RTMFP connection to **Adobe Chicago IL USA server**. It also indicates the behaviors of NAT and firewall filtering of the network where the client locates.

This checker does the following tests:

- *Knows public IP address of self.* This checks whether there is address translation.
- *Public UDP port number the same as local UDP port number.* This checks whether there is port translation. If there are neither address translation nor port translation, no NAT exists.
- *Can receive from same IP address, same UDP port number.* This checks whether NAT and firewall filtering is static. This answer should always be *YES* because it is mandatory for RTMFP establishing initial connection.
- *Can receive from same IP address, different UDP port number.* This checks whether NAT and firewall filtering is port restricted.
- *Can send to different IP address, different UDP port number.* This checks whether NAT and firewall filtering is address restricted.
- *Can send to different IP address after server introduction.* This test is like opening a new RTMFP connection after the initial connection. This answer should be *YES* if the initial connection can be made.
- *Source IP address is preserved from original connection.* This checks the type of NAT. It can be one of the following: a cone NAT, a symmetric NAT with single IP address, a symmetric NAT with multiple IP addresses but the same address happens to be used this time. If running checker repeatedly and the values change, NAT is symmetric and with multiple IP addresses.
- *Source UDP port number is preserved from original connection.* This checks whether an NAT is a cone NAT or a symmetric NAT.

[4]http://tools.ietf.org/wg/behave/
[5]http://cc.rtmfp.net

3.2.3 Peer-assisted Networking Setting of Flash Player

The demonstrator allows sharing video among an RTMFP group. Each member has to enable P2P uplink to share its bandwidth with others. So it is necessary to enable peer-assisted networking setting in Flash Player. Flash Player Setting Managers has a peer-assisted networking setting panel. Figure 3.6 shows this panel for Windows 7. Using this panel allows Flash Player to share bandwidth with user's permission for one website.

Figure 3.6: Peer-assisted networking setting panel of Flash Player.

There are two ways to enable this option. One is to allow the specified website to share bandwidth all the time without asking. For example, the blue line in Figure 3.6 has set its Network Access to "*Allow*". Another way is to set Network Access of a website with "*Ask me*". In this case, Flash Player will ask the user before sharing bandwidth using a pop-out window.

3.3 Demonstrator

This thesis is meant to implement a demonstrator that delivers live video to multiple clients using multicast. Our prototype demonstrator is built on Flash platform and is accessed using this link *http://fms.medianetworkservices.com/multicast_dem*
It consists of publisher, receiver, *multicast* application on Flash Media Server and Apache HTTP Server. Figure 3.7 shows an example of the working demonstrator with one receiver. For a video streaming, the demonstrator can only have one publisher, but multiple receivers are allowed.

The publisher and the receiver are two computers opening HTML pages *RTMP_Publusher.html* and *RTMFP_Receiver.html* respectively. These two pages

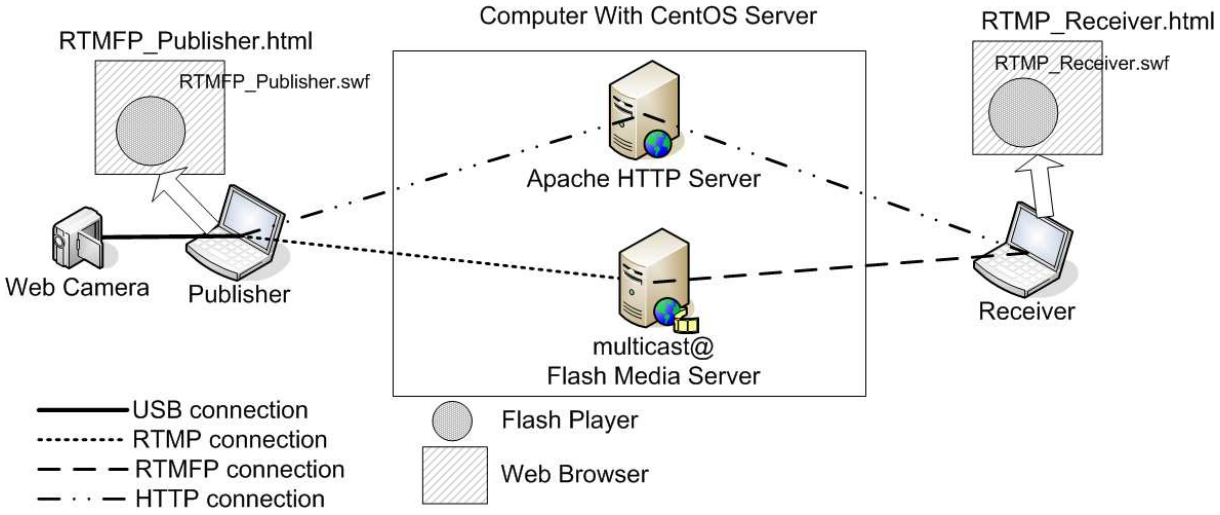


Figure 3.7: An example of the working demonstrator: it consists of a web camera, a publisher, a receiver, *multicast* applications on FMS and Apache HTTP Server. The computer opening *RTMP_Publusher.html* is named as a publisher. The publisher captures live video from web camera and sends this video over an RTMP connection. The computer opening *RTMFP_Receiver.html* is names as a receiver. The receiver obtains video over RTMFP connection, and then plays it. The *multicast* application is a server-side application on FMS. It receives video from the publisher and publishes this video to an RTMFP group. The Apache HTTP Server delivers *.html* and *.swf* files over HTTP connections.

wrap *RTMP_Publusher.swf* and *RTMFP_Receiver.swf* respectively. The SWF files are compiled Flash applications that are programmed using MXML and ActionScript 3.0 in Flash Builder. Their detailed implementations are presented in Section 3.3.2 and Section 3.3.3. The publisher connects to *multicast* application on FMS over an RTMP connection. It captures live video from the connected web camera and sends this video to *multicast* application on FMS. The receiver connects to *multicast* application on FMS over an RTMFP connection. It joins this RTFMP group, receives video published to this group and then plays it. The RTMFP group is specified using RTMFP group specifier described in Section 3.3.1.

The *multicast* application on FMS is a server-side application provided by Adobe. Its tasks are to receive video from the publisher and publish this video to an RTMFP group. A description of this application is presented in Section 3.3.4.

3.3.1 RTMFP Group Specifier

An RTMFP group can be defined using the *GroupSpecifier* class. The demonstrator allows the user to define the group with one of the following multicast types.

P2P multicast

The ActionScript codes to define parameters for a group using only P2P multicast are in Listing 3.3.

Listing 3.3: Parameters of an RTMFP group using only P2P multicast

```
GroupSpecifier groupspec = new GroupSpecifier(groupName);
groupspec.serverChannelEnabled = true;
groupspec.multicastEnabled = true;
groupspec.setPublishPassword(publishPwd);
groupspec.peerToPeerDisabled = false;
```

IP multicast

The ActionScript codes to define parameters for a group using only IP multicast are in Listing 3.4.

Listing 3.4: Parameters of an RTMFP group using only IP multicast

```
GroupSpecifier groupspec = new GroupSpecifier(groupName);
groupspec.serverChannelEnabled = true;
groupspec.multicastEnabled = true;
groupspec.setPublishPassword(publishPwd);
groupspec.peerToPeerDisabled = true;
groupspec.addIPMulticastAddress(multicastAddr);
groupspec.ipMulticastMemberUpdatesEnabled = true;
```

Fusion multicast

A group with fusion multicast indicates that video delivery can use IP multicast and P2P multicast coordinated and concurrently. IP multicast will be preferred. The ActionScript codes to define parameters for the group are in Listing 3.5.

Listing 3.5: Parameters of an RTMFP group using fusion multicast

```
GroupSpecifier groupspec = new GroupSpecifier(groupName);
groupspec.serverChannelEnabled = true;
groupspec.multicastEnabled = true;
groupspec.setPublishPassword(publishPwd);
groupspec.peerToPeerDisabled = false;
groupspec.addIPMulticastAddress(multicastAddr);
groupspec.ipMulticastMemberUpdatesEnabled = true;
```

3.3.2 Publisher

The publisher opens the HTML page *RTMP_Publisher.html* in a web browser. Its major tasks are to capture live video from camera and send this video to FMS over an RTMP connection.

GUI

When open the publisher on a web browser, its starting GUI is loaded as shown in Figure 3.8. It has a panel *Settings for Publisher*. On this panel, the user can specify the name or IP address of Flash Media Server and its server-side application that the publisher is connecting to. There are two tools to configure the video published. One tool is *Multicast Config Tool*. It is used to specify the parameters of an RTMFP group. These parameters include multicast type, stream name, group name, publish password, and IP multicast address if IP multicast is enabled. Another tool is *Video Quality Tool*. It is used to choosing published video's resolution. This demonstrator allows the user to choose video resolution among 160×120, 320×240, 640×480 and 1280×960 pixels.

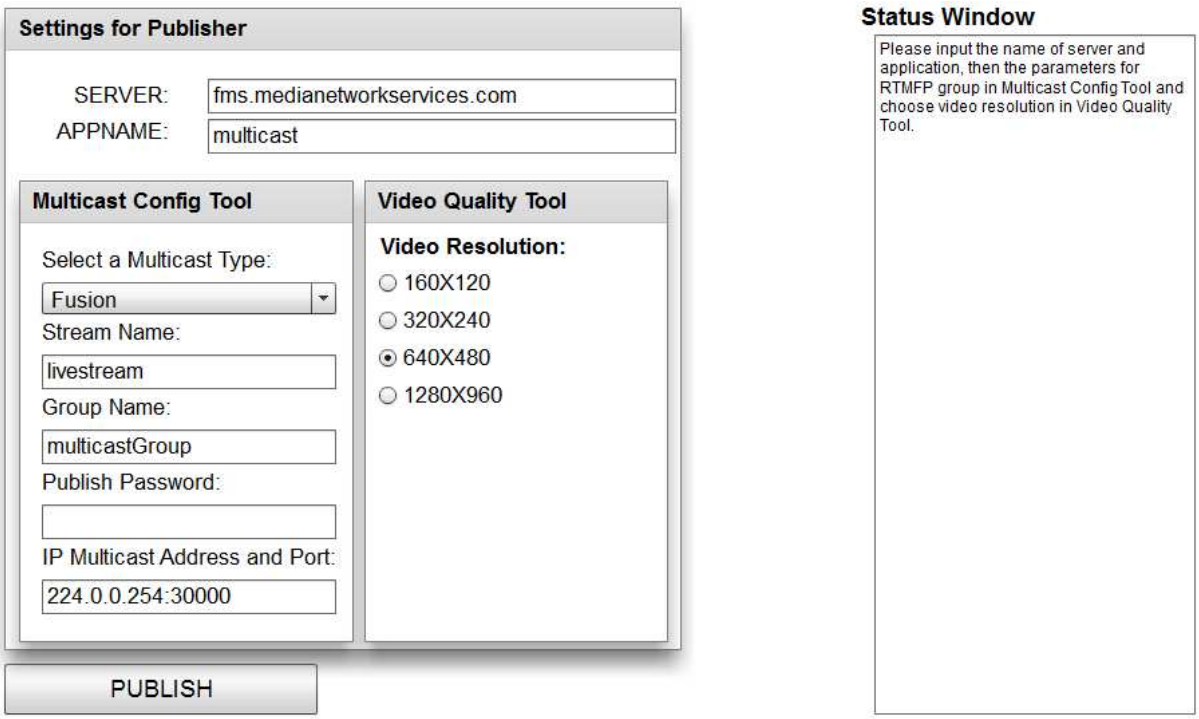


Figure 3.8: Starting GUI of publisher with *Settings for Publisher* panel.

The publisher starts video streaming by clicking *PUBLISH* button. The streaming GUI is loaded as shown in Figure 3.9. There is no *Settings for Publisher* panel for configuration. The video captured from camera is displayed. The user can click *STOP* button to stop video streaming and come back to the starting GUI.

In both GUIs, there is a *Status Window* that shows the texts specified in AcitonScript codes of publisher. This text window is used as debugging tool in our programming. These texts contain information about invoked NetStatusEvent events. A successful publishing as shown in 3.9 has invoked the following events in order:

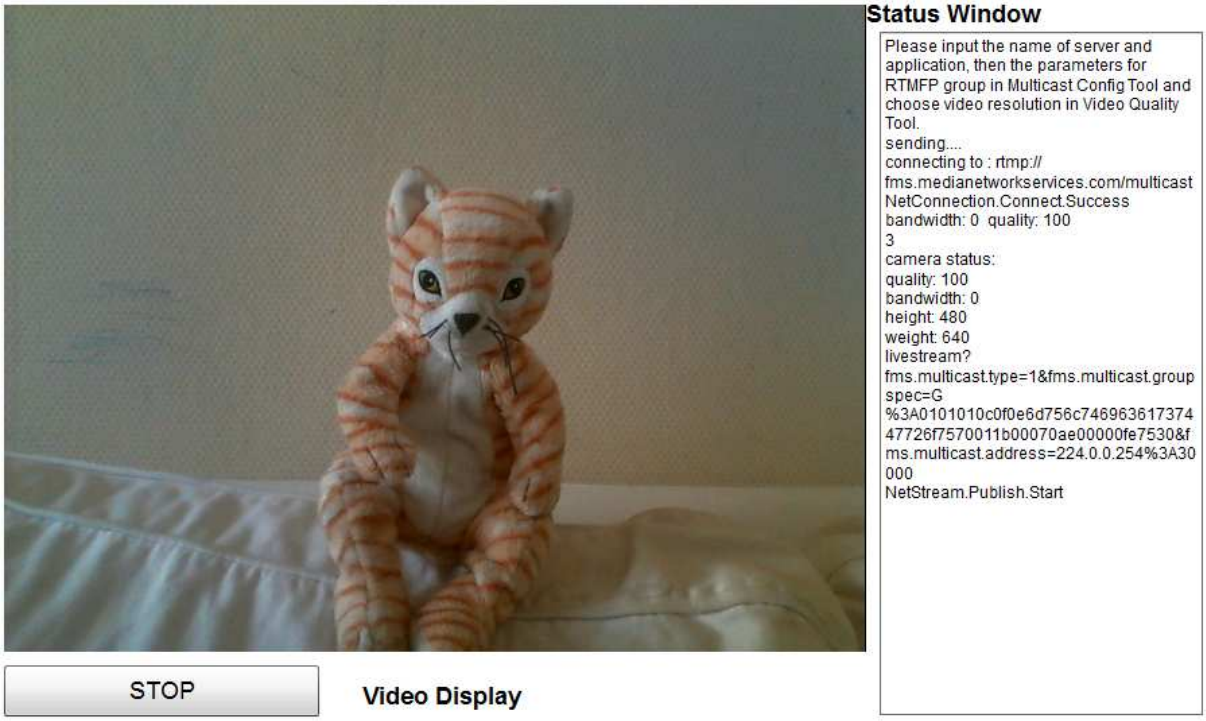


Figure 3.9: Streaming GUI of publisher with video captured from camera.

1. *NetConnection.Connect.Success*,

2. *NetStream.Publish.Start*.

The last event implies that video starts to be sent.

Camera setting

The *Camera* class allows the developer to set the camera capture mode. In this demonstrator, only video resolution can be changed on publisher's starting GUI. For other camera settings, the publisher uses the following default values:

- $fps = 15$ frames per second.

- $quality = 100$. This value means the highest picture quality and no compression being applied to each video frame.

- $bandwidth = 0$. This value indicates that quality takes precedence. The runtime uses as much bandwidth as required to maintain the specified quality. If necessary, the runtime reduces the frame rate to maintain picture quality. In general, as motion increases, bandwidth usage also increases.

ActionScript codes

The actions behind GUI are programmed using ActionScript 3.0 language. The scripts in this section are not the complete program used for the publisher. They are code fragments explaining the core operations of the publisher. By clicking *PUBLISH* button, the publisher first establishes an RTMP connection with *multicast* application at FMS using the scripts in Listing 3.6.

Listing 3.6: The publisher establishes an RTMP connection with a server-side application at FMS.

```
NetConnection netConnection = new NetConnection();
// Create an event listener to NetStatusEvent related to NetConnection.
netConnection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
netConnection.connect("rtmp://" + SERVER + "/" + APPNAME);
```

When *NetConnection.connect()* is called, the status of connection is reported as a *NetStatusEvent* object. If the information in the object contains *NetConnection.Connect.Success*, the publisher can start sending video captured from camera using the scripts in Listing 3.7.

Listing 3.7: The publisher starts sending video.

```
private function netStatusHandler(event:NetStatusEvent):void
{
  switch(event.info.code)
  {
    case "NetConncetion.Connect.Success":
      NetStream netStream = new NetStream(netConnection);
      // Attach camera to stream for sending.
      netStream.addCamera(camera);
      netStream.publish(publishName);
      break;
  }
}
```

This sending video will be re-published to an RTMFP group by *multicast* application on FMS. The group is specified by the publisher. The information of group is sent in the name of sent stream *publishName* in Listing 3.7. The format of *publishName* is listed in Listing 3.8.

Listing 3.8: The name of sent stream from the publisher.

```
var publishName:String = streamName +
      "?fms.multicast.type=multicastType" +
      "&fms.multicast.groupspec=" +
      escape(groupspec.groupspecWithAuthorizations()) +
      "&fms.multicast.address=" + // only if IP multicast is enabled.
      escape(multicastAddr);
```

The string *streamName* is the name of stream published to RTMFP group. The string *multicastType* has value among 1, 2 and 3 that represent respectively Fusion, IP and P2P multicast. The string generated by *escape(groupspec.groupspecWithA*

describes the parameters of RTMFP group. The *groupspec* is a GroupSpecifier object defined as Section 3.3.1. If IP multicast is enabled, the string from *escape(multicastAddress)* contains IP multicast group address and port number.

3.3.3 Receiver

The receiver opens the HTML page *RTMFP_Receiver.html* in a web browser. Its major tasks are to receive the video published for an RTMFP group and play it.

GUI

When open the receiver on a web browser, its starting GUI is loaded as shown in Figure 3.10. It has a *Settings for Receiver* panel. On this panel, the user can specify the name or IP address of Flash Media Server and its server-side application that the receiver is connecting to. There is a *Multicast Config Tool*. Similar to the tool in publisher, it is used to specify the parameters of an RTMFP group. To obtain video published, the group specification shall be the same as that in the publisher.

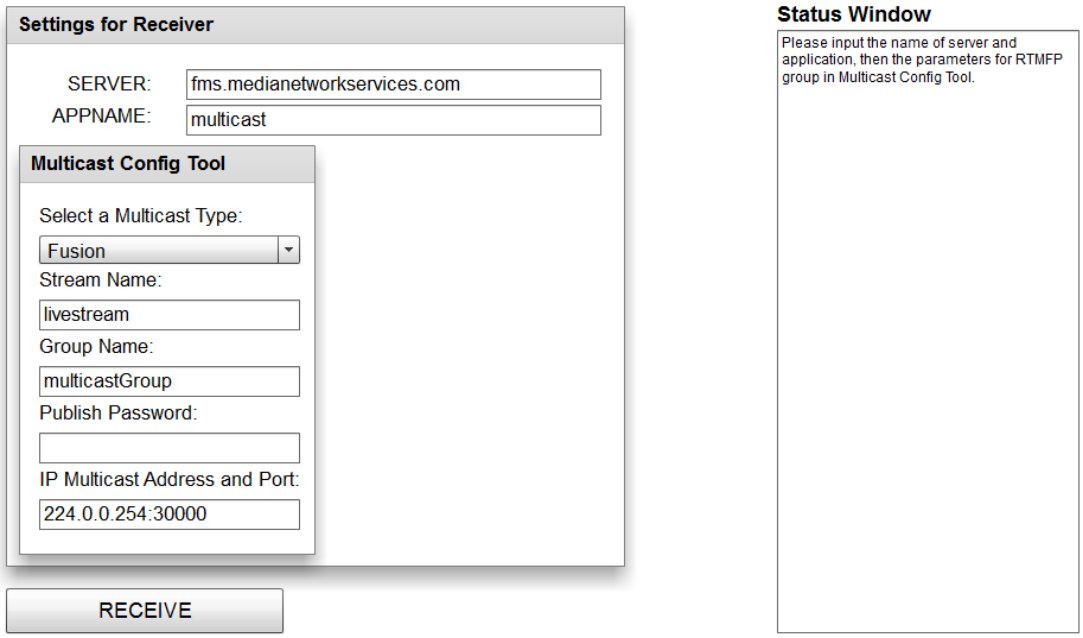


Figure 3.10: Starting GUI of receiver with *Settings for Receiver* panel.

The receiver starts video streaming by clicking *RECEIVE* button. The streaming GUI is loaded as shown in Figure 3.11. There is no *Settings for Receiver* panel. The obtained video is displayed. The receiving and sending transmission rates are illustrated by the line plots. The rates include media data and control overhead rates at this receiver. The user can click *STOP* button to stop video streaming and come back to the starting GUI.

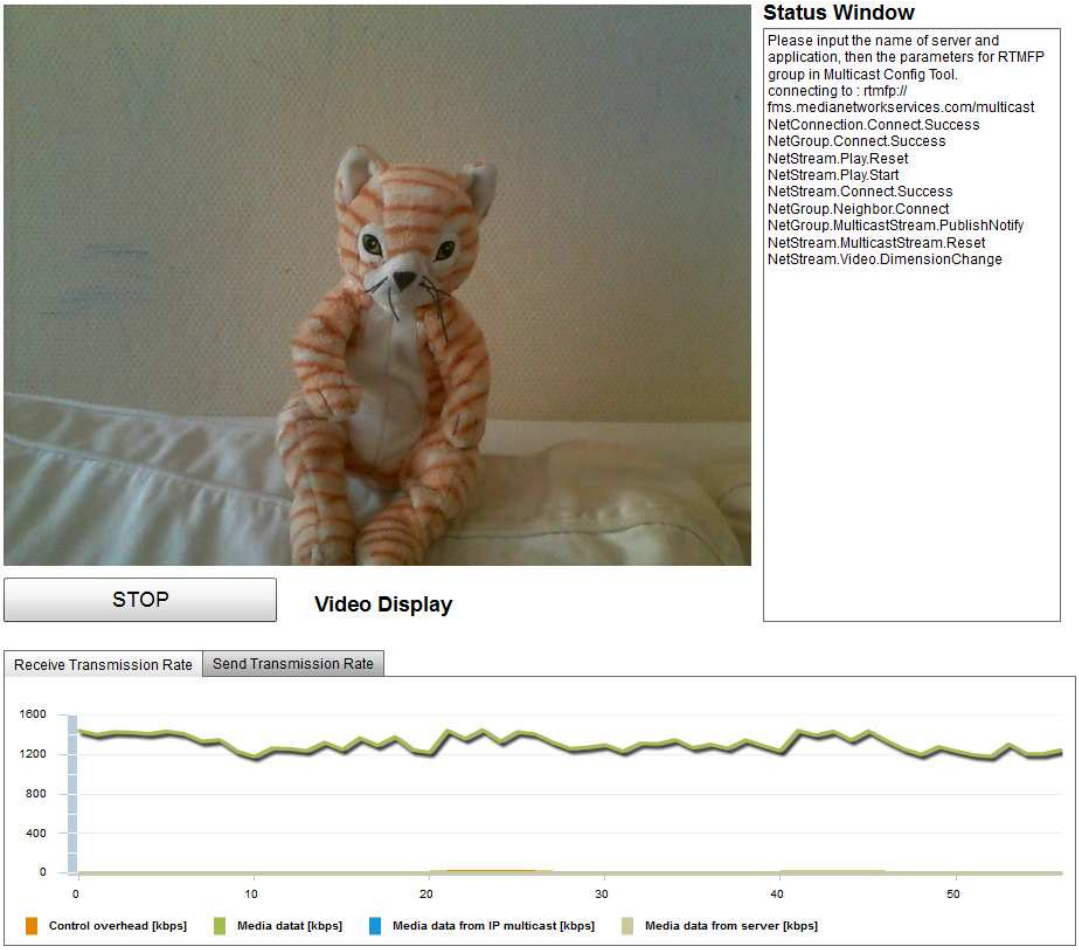


Figure 3.11: Streaming GUI of receiver with *Transmission Rate* charts.

As the publisher, both GUIs contain a *Status Window*. A successful receiving as shown in Figure 3.11 has invoked the following NetStatusEvent events in order:

1. *NetConnection.Connect.Success*,
2. *NetGroup.Connect.Success*,
3. *NetStream.Play.Reset*,
4. *NetStream.Play.Start*,
5. *NetStream.Connect.Success*,
6. *NetGroup.Neighbor.Connect*,
7. *NetGroup.MulticastStream.PublishNotify*,
8. *NetStream.MulticastStream.Reset*,
9. *NetStream.Video.DimensionChange*.

The receiver in this demonstrator obtains video only by multicast communication, i.e. as a multicast stream. The video starts to play after the multicast stream has been received.

ActionScript codes

As for the publisher, this section introduces some code fragments explaining the core operations of the receiver. By clicking *RECEIVE* button, the receiver first establishes an RTMFP connection with *multicast* application at FMS using the scripts in Listing 3.9.

Listing 3.9: The receiver establishes an RTMFP connection with a server-side application at FMS

```
NetConnection netConnection = new NetConnection();
// Create an event listener to NetStatusEvent related to NetConnection.
netConnection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
netConnection.connect("rtmfp://" + SEVER + "/" + APPNAME);
```

When *NetConnection.connect()* is called, the status of connection is reported as a *NetStatusEvent* object. If the information in the object contains *NetConnection.Connect.Success*, the receiver joins the RTMFP group using the scripts in Listing 3.10. The group is described with a string *groupspec*. This string is created using *GroupSpecifier.groupspecWithAuthorizations()*.

Listing 3.10: The receivers joins the RTMFP group.

```
private function netStatusHandler(event:NetStatusEvent):void
{
  switch(event.info.code)
  {
  case "NetConncetion.Connect.Success":
    NetGroup netGroup = new NetGroup(netConnection, groupspec);
    // Create an event listener to NetStatusEvent related to the status of NetGroup.
    netGroup.addEventListener(NetStatusEvent.NET_STATUS,
    netStatusHandler);
    break;
  }
}
```

When *NetGroup* is created on *NetConnection* and associated with the *groupspec* string, the status of *NetGroup* is reported as a *NetStatusEvent* object. If the information in the object contains *NetGroup.Connect.Success*, the receiver starts to receive and play stream published to the group using the scripts in Listing 3.11. *NetStream* is created on *NetConnection* and associated with the *groupspec* string. It indicates this stream is related to the specified RTMFP group.

Listing 3.11: The receiver receives and plays the stream published to the RTMFP group.

```
private function netStatusHandler(event:NetStatusEvent):void
```

```
{
        switch(event.info.code)
        {
        case "NetGroup.Connect.Success":
                NetStream netStream = new NetStream(netConnection, groupspec);
                netStream.play(streamName);
                break;
        }
}
```

3.3.4 Server-side application

The server-side application in the demonstrator is the sample application *multicast* located in the *rootinstall/applications/multicast* folder. This application is available with default installation of FMS. This folder consists of a configuration file *Application.xml* and an ActionScript file *main.asc* as shown in Figure 3.4(a).

Application.xml configures some settings for this application and overrides the corresponding settings in *Application.xml* in the virtual host level. One of important settings is to turn off live queuing and aggregate messages. This can reduce delay in video's playback.

main.asc is an ActionScript file written in Adobe's Server-Side ActionScript language. It accepts published stream over an RTMP conneciton, and then republishes this stream to an RTMFP group and/or an IP multicast group. The RTMFP group specification and the IP multicast group address are obtained from the stream name of published video as listed in Listing 3.8. When Publisher connects to this application and publishes a stream, *main.asc* handles the received stream as illustrated in Figure 3.12.

3.4 Summary

In this chapter, we have implemented a prototype demonstrator on Adobe's Flash platform. It delivers captured live video to multiple clients using multicast communications. At present, this demonstrator is available at this link *http://fms.medianetworkservices.com/multicast_demonstrator/*. In next chapter, a series of experiments using this demonstrator in Internet are performed. Internet traffic during experimenting are measured and analyzed.

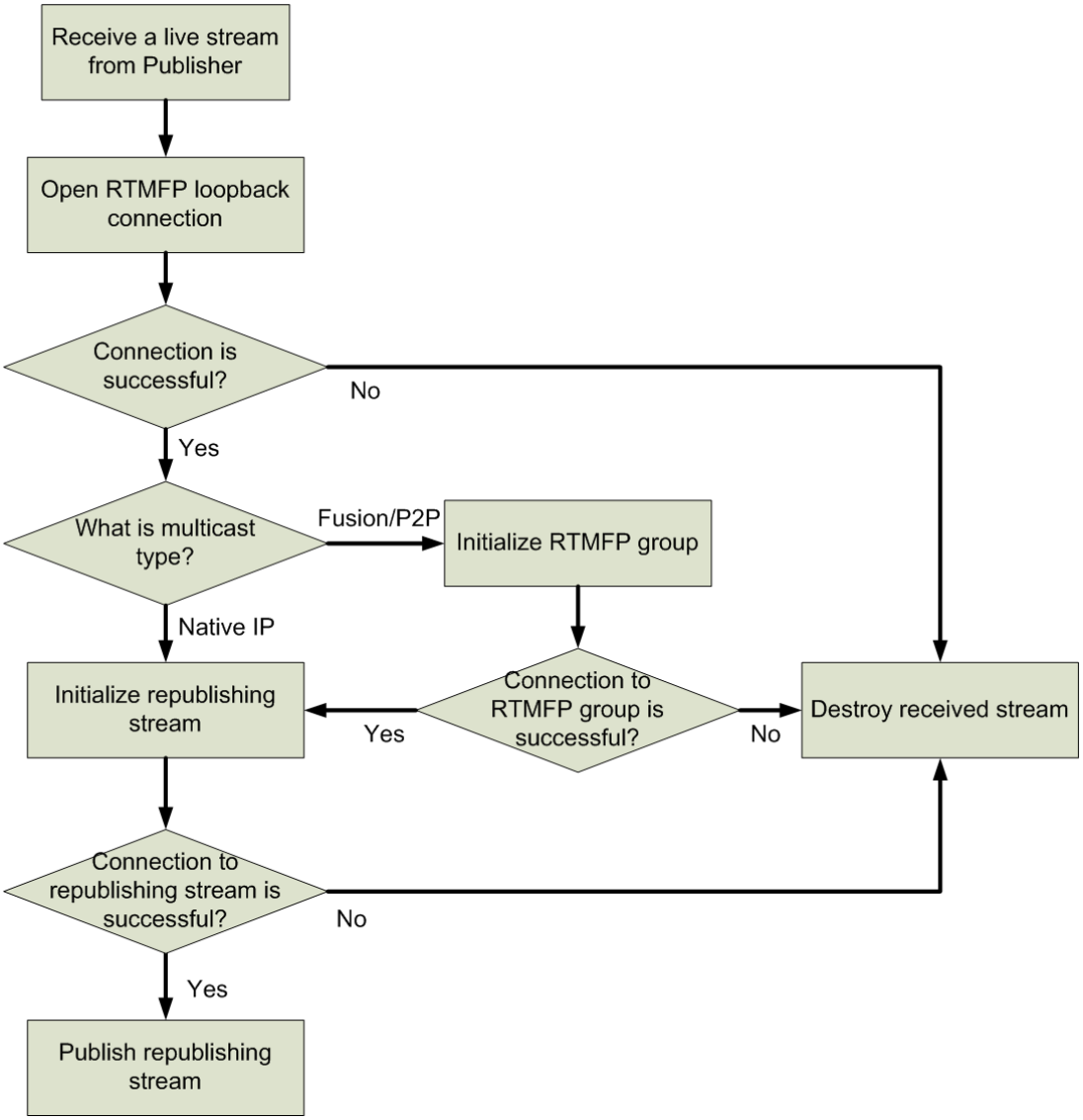


Figure 3.12: Diagram of how server-side script republishes stream to an RTMFP group.

Chapter 4

Measurements and Results

The demonstrator has been tested in several measurement scenarios. The measurements and results with only running publisher or receivers are described in section 4.1. The measurements and results with running publisher and receivers are described in section 4.2

4.1 Measurements of Demonstrator's Components

4.1.1 Only Publisher

The publisher in the demonstrator captures video from camera into a NetStream object and deliver it to FMS over an RTMP connection. In this measurement scenario, the publisher is run on a computer located in home network. This computer is connected to the Internet via a wireless access point. The camera connected to this computer is Creative Live! Cam Optia Pro VF0380 Web Camera. Record the packets that are captured using Wireshark 1.6.1 with different video resolutions. The resolutions can be selected on the publisher's GUI.

RTMP packets

Wireshark can recognize the RTMP packets. The publisher establishes an RTMP connection and starts streaming as the procedure illustrated in Figure 4.1.

Once the RTMP connection is established, the publisher begins to deliver the RTMP Video Data. An RTMP packet that contains Video Data consists of an RTMP header and an RTMP body as illustrated in Figure 4.2. The size of RTMP header is 8 bytes. The RTMP body consists of 1 byte control and video data. The control byte indicates the type and format of video. In this measurement, it indicates that the video encoding in Flash Player uses Sorenson H.263, also known as Sorensen Spark. The Sorensen H.263 is one of three video compression formats for Flash video. The other two are H.264 and VP6.

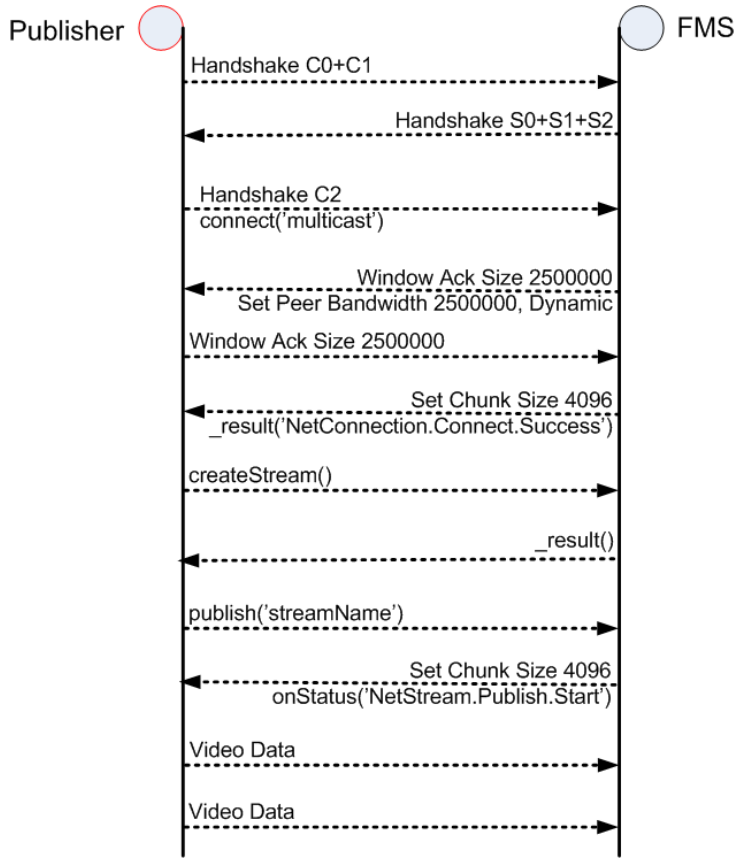


Figure 4.1: The procedure of RTMP connection establishment and streaming from the publisher: it establishes an RTMP connection between the publisher and FMS using Handshake RTMP packets at first. Then the publisher publishes the stream and delivers the video data over the RTMP connection.

The Sorenson H.263 is an incomplete implementation of H.263. The video data has three types: keyframe, inter-frame, disposable inter-frame.

Packets transmission

The connection between the publisher and FMS is a TCP connection. By observing the packets transmitted from the publisher to FMS, there are also TCP packets in addition to the RTMP packets. A segment of the packets from the publisher to FMS is illustrated in Figure 4.3. The TCP packets are marked with the red boxes. These packets have only two sizes: 1514 and 1230 bytes. These packets contain also video data according to the common sense. The RTMP's specification[19] also informs that there is a kind of RTMP packets without message header. This kind of packets belong to the same stream from the previous RTMP Video Data.

The packets from FMS to the publisher are all TCP packets. These packets are all Acknowledges for video data segments. However, not all frames that contain video data are acknowledged. A segment of the packets delivered between the publisher and FMS is illustrated in Figure 4.4. In this example,

RTMP Header

Format	Chunk Stream ID	Timestamp delta	Timestamp	Body size	Type ID (Video Data)

RTMP Body

Video type	Video Format (Sorensen H.263)	Video data

Figure 4.2: An RTMP packet that contains Video Data consists of an RTMP header and an RTMP body.

the TCP packets marked in the red boxes have no acknowledge for their video segments.

Packet size distribution

The publisher can deliver video with different resolutions. In this measurement scenario, record the packets delivered from the publisher under the respective video resolutions. Calculate the cumulative fraction for each packet size. The packet size is the TCP payload length of each packet. Figure 4.5 illustrates the packet size distributions with four kinds of video resolutions. The packets used in calculations are all TCP packets delivered from the publisher to FMS within one minute from the beginning of connection establishment.

The plots in Figure 4.5 shows that the TCP packets with fixed size 1176 and 1460 bytes have a significant proportion of all packets delivered from the publisher. The proportion of these TCP packets is increased with increasing video resolution as illustrated in Table 4.1. However, this proportion does not increase much when the video resolution is larger than 480×360 pixels.

Video Resolution	**(%) packets(1176 bytes)**	**(%)packets(1460 bytes)**
160×120	7.8	32.0
320×240	11.9	51.9
480×360	26.4	62.5
800×480	28.2	64.1

Table 4.1: Percent of TCP packets with fixed size delivered from the publisher.

Based on these captured packets, we can calculate the transmission bandwidth of the publisher as listed in Table 4.2. The calculation uses Equation 4.1. The publisher delivers the live video using the command *NetStream.publish('streamName')*. It indicates the most transmitted packets are video data. According to our observation, there are packets that don't contain video data, such as packets

No.	Time	Source	Destination	Protocol	Length	Info
2267	21.207370	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=294215 Ack=3582 Win=16856 Len=1460
2268	21.207384	192.168.0.101	93.93.96.91	RTMP	261	Video Data
2269	21.239250	192.168.0.101	93.93.96.91	RTMP	1471	Video Data
2272	21.309913	192.168.0.101	93.93.96.91	RTMP	1430	Video Data
2274	21.407082	192.168.0.101	93.93.96.91	RTMP	1459	Video Data
2275	21.438921	192.168.0.101	93.93.96.91	RTMP	995	Video Data
2278	21.511364	192.168.0.101	93.93.96.91	RTMP	1317	Video Data
2280	21.605961	192.168.0.101	93.93.96.91	RTMP	1210	Video Data
2281	21.639043	192.168.0.101	93.93.96.91	RTMP	1233	Video Data
2323	21.711034	192.168.0.101	93.93.96.91	RTMP	1033	Video Data
2324	21.805978	192.168.0.101	93.93.96.91	RTMP	1240	Video Data
2328	21.910656	192.168.0.101	93.93.96.91	RTMP	592	Video Data
2329	21.942842	192.168.0.101	93.93.96.91	RTMP	1134	Video Data
2332	22.009316	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=308402 Ack=3582 Win=16856 Len=1460
2333	22.009327	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=309862 Ack=3582 Win=16856 Len=1460
2334	22.009335	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=311322 Ack=3582 Win=16856 Len=1176
2335	22.009680	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=312498 Ack=3582 Win=16856 Len=1460
2336	22.009690	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=313958 Ack=3582 Win=16856 Len=1460
2337	22.009695	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=315418 Ack=3582 Win=16856 Len=1176
2338	22.009909	192.168.0.101	93.93.96.91	RTMP	528	Video Data
2365	22.111364	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=317068 Ack=3582 Win=16856 Len=1460
2366	22.111375	192.168.0.101	93.93.96.91	RTMP	124	Video Data
2386	22.143127	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=318598 Ack=3582 Win=16856 Len=1460
2387	22.143138	192.168.0.101	93.93.96.91	RTMP	202	Video Data
2391	22.207067	192.168.0.101	93.93.96.91	RTMP	1328	Video Data
2395	22.311047	192.168.0.101	93.93.96.91	RTMP	1459	Video Data
2396	22.342928	192.168.0.101	93.93.96.91	RTMP	1046	Video Data
2398	22.407262	192.168.0.101	93.93.96.91	RTMP	1358	Video Data
2401	22.509800	192.168.0.101	93.93.96.91	RTMP	1286	Video Data
2402	22.543121	192.168.0.101	93.93.96.91	RTMP	1418	Video Data
2404	22.606897	192.168.0.101	93.93.96.91	RTMP	1210	Video Data
2447	22.711111	192.168.0.101	93.93.96.91	RTMP	1369	Video Data
2448	22.742913	192.168.0.101	93.93.96.91	RTMP	768	Video Data
2450	22.806973	192.168.0.101	93.93.96.91	RTMP	1151	Video Data
2454	22.910110	192.168.0.101	93.93.96.91	RTMP	1208	Video Data
2456	23.006983	192.168.0.101	93.93.96.91	RTMP	1177	Video Data
2476	23.041490	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=334336 Ack=3582 Win=16856 Len=1460
2477	23.041516	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=335796 Ack=3582 Win=16856 Len=1460
2478	23.041521	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=337256 Ack=3582 Win=16856 Len=1176
2479	23.041869	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=338432 Ack=3582 Win=16856 Len=1460
2480	23.041878	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=339892 Ack=3582 Win=16856 Len=1460
2481	23.041884	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=341352 Ack=3582 Win=16856 Len=1176
2482	23.042102	192.168.0.101	93.93.96.91	RTMP	678	Video Data
2504	23.111322	192.168.0.101	93.93.96.91	RTMP	1207	Video Data
2513	23.206720	192.168.0.101	93.93.96.91	RTMP	1420	Video Data
2514	23.238814	192.168.0.101	93.93.96.91	RTMP	988	Video Data

Figure 4.3: A segment of the packets delivered from the publisher to FMS.

for connection establishment. Moreover, the RTMP packet for Video Data contains other data than video data, The other data has only 9 bytes that is much smaller than the size of video data in the RTMP body. Therefore, the calculated bandwidth here can be considered to be close to the actual video transmission bandwidth of the publisher.

$$bandwidth = \frac{\sum_{allpackets} TCPpayload(bytes) \times 8}{60 \times 1000} \tag{4.1}$$

Video Resolution	160×120	320×240	480×360	800×600
Bandwidth (kbps)	183	345	930	1017

Table 4.2: Transmission bandwidth from the publisher.

4.1.2 Only Receivers

In this measurement scenario, only receivers are run on two computers located in home network. The receivers are in the same RTMFP group and no stream is published to this group on FMS. In the beginning, only one receiver is connected to FMS and joins the specified RTMFP group. This group has only one member. The second receiver connects to FMS and joins this group after

No.	Time	Source	Destination	Protocol	Length	Info
4277	37.474938	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=657224 Win=248064 Len=0
4278	37.511137	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=657224 Ack=3582 Win=16856 Len=1460
4279	37.511146	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=658684 Ack=3582 Win=16856 Len=1460
4280	37.511149	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=660144 Ack=3582 Win=16856 Len=1176
4281	37.511353	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=661320 Ack=3582 Win=16856 Len=1460
4282	37.511358	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=662780 Ack=3582 Win=16856 Len=1460
4283	37.511360	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=664240 Ack=3582 Win=16856 Len=1176
4284	37.511513	192.168.0.101	93.93.96.91	RTMP	635	Video Data
4285	37.577211	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=660144 Win=248064 Len=0
4286	37.580357	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=661320 Win=248064 Len=0
4287	37.580814	192.168.0.101	93.93.96.91	RTMP	962	Video Data
4288	37.584712	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=665416 Win=248064 Len=0
4289	37.588029	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=665997 Win=248064 Len=0
4291	37.606758	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=666905 Ack=3582 Win=16856 Len=1460
4292	37.606769	192.168.0.101	93.93.96.91	RTMP	195	Video Data
4331	37.647657	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=666905 Win=248064 Len=0
4332	37.672853	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=668506 Win=248064 Len=0
4333	37.711164	192.168.0.101	93.93.96.91	RTMP	1506	Video Data
4334	37.743030	192.168.0.101	93.93.96.91	RTMP	1415	Video Data
4335	37.777076	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=669958 Win=248064 Len=0
4336	37.806927	192.168.0.101	93.93.96.91	RTMP	1183	Video Data
4337	37.808484	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=671319 Win=248064 Len=0
4338	37.878209	93.93.96.91	192.168.0.101	TCP	54	macromedia-fcs > 57350 [ACK] Seq=3582 Ack=672448 Win=248064 Len=0

Figure 4.4: A segment of the packets delivered between the publisher and FMS.

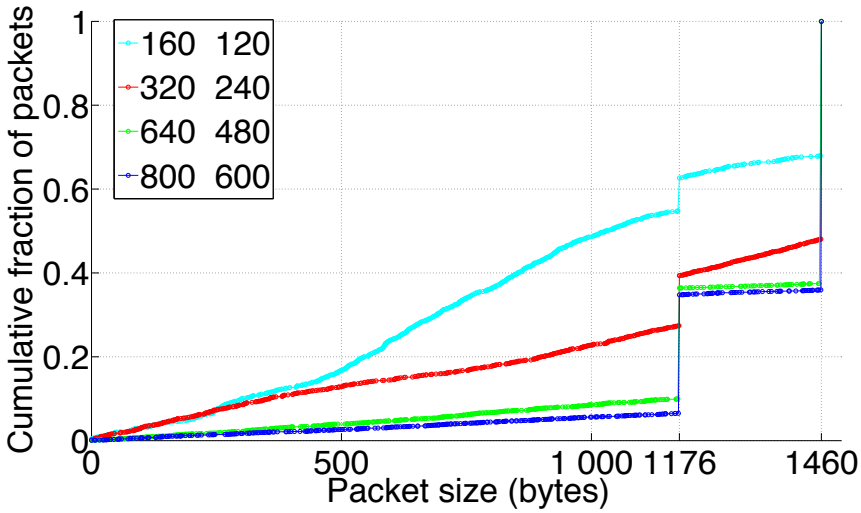


Figure 4.5: The cumulative fraction of packets from the publisher to FMS within one minute with different video resolutions.

a while. Now the group has two members. Then the second receiver is closed and this indicates a member has left the group. The packets are recorded on first receiver using Wireshark 1.6.1.

New RTMFP group

When a receiver is connected to FMS, an RTMFP group with only one member is created. A sample of the communication between the receiver and FMS since the beginning is illustrated in Figure 4.6. In this sample, the communication consists of three stages. In the first stage, the receiver sends an RTMFP request to port 1935 on FMS and FMS sends a response back to the receiver. The response contains the information about the RTMFP redirect port number for next stage. In the second stage, the receiver communicates with FMS over port 19351 in order to redirect to another port for session establishment. However, there is no peer available on FMS in this case. The receiver will just exchange the information with FMS over port 19351 periodically. These periodical exchanges are inside stage 3. The stages 1 and 2 happen very fast, for

example 0.31 seconds in this sample. The period for exchanging in stage 3 is around 15 seconds. The exchanged packets have fixed-size 20 bytes, but their contents change over time.

No.	Time	Source	Destination	Protocol	Length	Info	
38	3.461389	192.168.0.101	93.93.96.91	UDP	126	Source port: 64545 Destination port: macromedia-fcs	Stage 1
39	3.513554	93.93.96.91	192.168.0.101	UDP	78	Source port: macromedia-fcs Destination port: 64545	
40	3.513794	192.168.0.101	93.93.96.91	UDP	126	Source port: 64545 Destination port: 19351	
42	3.566659	93.93.96.91	192.168.0.101	UDP	222	Source port: 19351 Destination port: 64545	
43	3.567350	192.168.0.101	93.93.96.91	UDP	350	Source port: 64545 Destination port: 19351	
44	3.618519	93.93.96.91	192.168.0.101	UDP	206	Source port: 19351 Destination port: 64545	
45	3.621456	192.168.0.101	93.93.96.91	UDP	350	Source port: 64545 Destination port: 19351	
46	3.671232	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
47	3.672692	93.93.96.91	192.168.0.101	UDP	94	Source port: 19351 Destination port: 64545	Stage 2
48	3.672886	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
49	3.673003	93.93.96.91	192.168.0.101	UDP	334	Source port: 19351 Destination port: 64545	
50	3.673201	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
51	3.685704	192.168.0.101	93.93.96.91	UDP	110	Source port: 64545 Destination port: 19351	
52	3.725758	192.168.0.101	93.93.96.91	UDP	142	Source port: 64545 Destination port: 19351	
53	3.736176	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
54	3.774537	93.93.96.91	192.168.0.101	UDP	94	Source port: 19351 Destination port: 64545	
55	3.774757	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
72	18.775183	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
73	18.824684	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
101	33.774707	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
102	33.824869	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
121	48.775391	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
122	48.824883	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
140	63.774942	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
141	63.824739	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
176	78.774601	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
177	78.823525	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	Stage 3
241	93.775195	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
242	93.823241	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
263	108.774811	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
264	108.823298	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
282	123.775396	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
283	123.824134	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
301	138.775020	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
302	138.828857	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	
330	153.774723	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351	
331	153.823911	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545	

Figure 4.6: The communication between the receiver and FMS when there is no published stream.

Join of a new member

When another receiver connects to FMS with the same groupspecifier as first receiver, this receiver joins the created RTMFP group as a new member and connects to first receiver over an RTMFP connection. Figure 4.7 illustrates the packet traffic on first receiver during joining group and session establishment procedure. First receiver acquires information of joining member by listening on redirect port 19351 on FMS as illustrated by the first arrow in Figure 4.7. This information contains IP address and port number of new member. In this measurement scenario, first receiver can find new member by exchanging packets with server from Internet Service Provider and router in home network. It begins to establish session with new member since second arrow illustrated in Figure 4.7. There is no streaming now. Two connected members exchange fixed-sized packets once session is established successfully. The sizes of packets are 164 and 20.

Leave of a member

When second receiver disconnects on purpose or because of failure, first receiver will recognize and confirm this leaving. In this measurement scenario, first receiver is remaining member and second receiver is leaving member of

No.	Time	Source	Destination	Protocol	Length	Info
380	183.824316	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545
437	198.775434	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351
438	198.824662	93.93.96.91	192.168.0.101	UDP	62	Source port: 19351 Destination port: 64545
439	207.610308	93.93.96.91	192.168.0.101	UDP	110	Source port: 19351 Destination port: 64545 ← new member
440	207.611126	192.168.0.101	93.93.96.91	UDP	62	Source port: 64545 Destination port: 19351
441	207.611215	192.168.0.101	93.93.96.91	UDP	110	Source port: 64545 Destination port: 19351
442	207.660896	93.93.96.91	192.168.0.101	UDP	126	Source port: 19351 Destination port: 64545
443	207.661147	192.168.0.101	95.34.117.249	UDP	110	Source port: 64545 Destination port: jstel
444	207.661211	192.168.0.101	192.168.201.1	UDP	110	Source port: 64545 Destination port: 53792
445	207.661266	192.168.0.101	192.168.194.1	UDP	110	Source port: 64545 Destination port: 53792
447	207.665132	95.34.117.249	192.168.0.101	ICMP	138	Destination unreachable (Port unreachable)
449	207.720980	192.168.0.101	192.168.0.102	UDP	110	Source port: 64545 Destination port: 53792 ← start session establishment
452	208.186492	192.168.0.102	192.168.0.101	UDP	270	Source port: 53792 Destination port: 64545
453	208.186910	192.168.0.101	192.168.0.102	UDP	350	Source port: 64545 Destination port: 53792
454	208.190608	192.168.0.102	192.168.0.101	UDP	142	Source port: 53792 Destination port: 64545
455	208.191736	192.168.0.101	192.168.0.102	UDP	206	Source port: 64545 Destination port: 53792
456	208.193076	192.168.0.102	192.168.0.101	UDP	62	Source port: 53792 Destination port: 64545
457	208.193395	192.168.0.102	192.168.0.101	UDP	222	Source port: 53792 Destination port: 64545
458	208.193512	192.168.0.101	192.168.0.102	UDP	142	Source port: 64545 Destination port: 53792
459	208.193554	192.168.0.101	192.168.0.102	UDP	78	Source port: 64545 Destination port: 53792
460	208.195012	192.168.0.102	192.168.0.101	UDP	126	Source port: 53792 Destination port: 64545
461	208.394842	192.168.0.101	192.168.0.102	UDP	62	Source port: 64545 Destination port: 53792
462	208.396125	192.168.0.102	192.168.0.101	UDP	62	Source port: 53792 Destination port: 64545
499	216.865240	192.168.0.101	192.168.0.102	UDP	190	Source port: 64545 Destination port: 53792
500	216.868052	192.168.0.102	192.168.0.101	UDP	206	Source port: 53792 Destination port: 64545
501	217.068144	192.168.0.101	192.168.0.102	UDP	62	Source port: 64545 Destination port: 53792
518	222.402252	192.168.0.102	192.168.0.101	UDP	190	Source port: 53792 Destination port: 64545
519	222.402494	192.168.0.101	192.168.0.102	UDP	206	Source port: 64545 Destination port: 53792

Figure 4.7: The procedure of a new member joining RTMFP group.

the RTMFP group. The remaining member concludes the leaving of a member by many times of checking. In each checking, the remaining member sends a message to IP address and port number that the leaving member has used in the session. The remaining member can not get reply over that IP address and port number because that member has left in this case. As it has waited for reply for ca. 5 seconds, the remaining member contacts FMS over the redirector port 19351 and tries to redirect to another member again. In this measurement scenario, the checking happens 11 times and the intervals between checks becomes longer for later checks as illustrated in Figure 4.8.

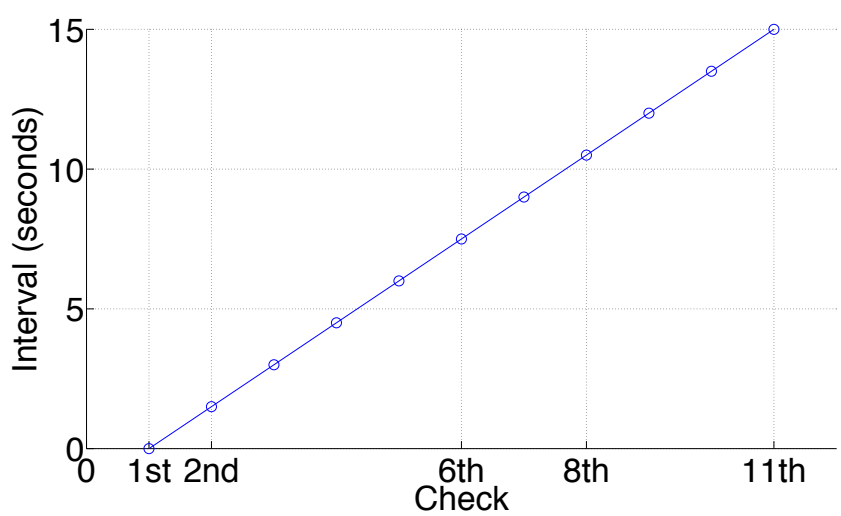


Figure 4.8: The checks and their time interval when a member leaves RTMFP group. The interval between two checks becomes longer for later checks.

4.2 Measurements of Streaming Demonstrator

This section presents three measurement scenarios in which one or more receivers play published video. The publisher and receivers are run on computers located in different network environments.

4.2.1 Case One: One Receiver

In this measurement scenario, A publisher and a receiver are running on two computers located on home network. A *multicast* application is running on computer located on MSNBONE.NET. This network is multicast-enabled. This measurement scenario is illustrated in Figure 4.9. The RTMFP group using P2P multicast has two members: FMS and the receiver. The publisher sends live video with different video resolution selected from publisher's GUI. The publisher and the receiver are in home network. Record all packets on receiver using Wireshark 1.6.1.

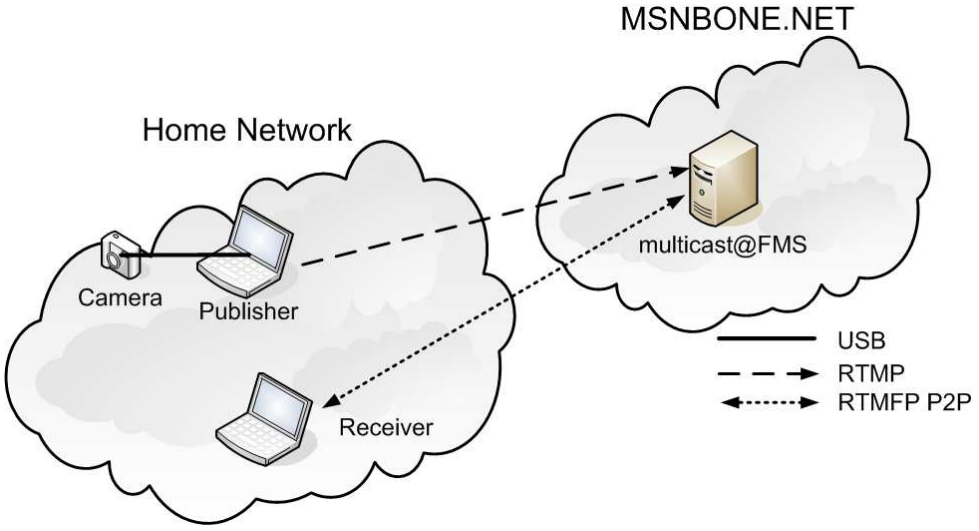


Figure 4.9: Measurement scenario for case one: only one receiver. A publisher and a receiver are running on two computers located on home network. A *multicast* application is running on computer located on MSNBONE.NET. This network is multicast-enabled.

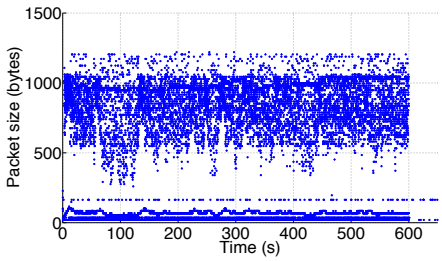
Session establishment with FMS

A session shall be established in order to deliver data between two peers. The multicast application on FMS creates a loopback RTMFP connection in order to establish session with the receivers connected. So there are two RTMFP group members in this measurement scenario. The receiver first listens on port 1935 and then redirect port 19351 on FMS. FMS allocates an port that the receiver can establish session with. Once the receiver acquires the port over redirect port 19351, it begins a session establishment. Even establishment

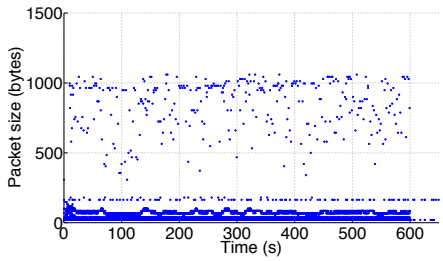
succeeds, the receiver also listens on redirect port 19351 periodically. The period is around 15 seconds. In this measurement scenario, the RTMFP group has two active members: FMS and receiver.

Packet distribution over time

A session is established between FMS and receiver in this measurement scenario. There are packets traveling inside this two-way channel. This section presents packet distribution over time for different published video resolutions as illustrated in Figure 4.10, Figure 4.11, Figure 4.12 and Figure 4.13. The packet size is the length of UDP packet payload in bytes. It is impossible to recognize the structure of each packet even it has been partly released in [41]. These packets captured on network interface are encrypted using Adobe's Cryptography Profile. They are decrypted by Flash Player.

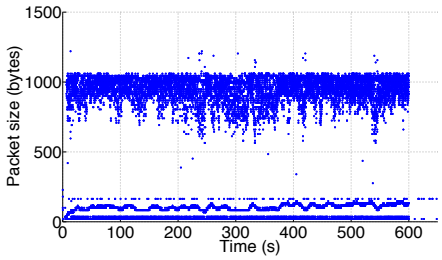


(a) Received packets vs. Time.

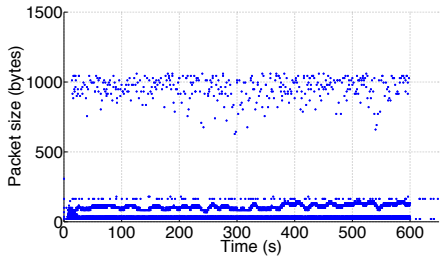


(b) Sent packets vs. Time.

Figure 4.10: Video resolution 160×120: packet distribution over time on one receiver for case one.



(a) Received packets vs. Time.



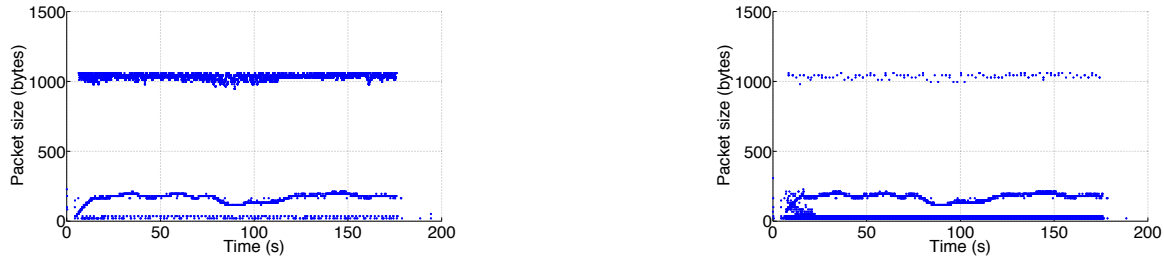
(b) Sent packets vs. Time.

Figure 4.11: Video resolution 320×240: packet distribution over time on one receiver for case one.

The size of all packets is between 20 and 1060 bytes in these measurements. The payloads can contain controlling data or video data. The plots in this section show that the packets can be categorized into two groups according to their size: small packet group and large packet group. In Figure 4.10 and Figure 4.11, the packets larger than 164 bytes are in large packet group. In Figure 4.12 and Figure 4.13, the packets larger than 228 bytes are in large

(a) Received packets vs. Time. (b) Sent packets vs. Time.

Figure 4.12: Video resolution 640×480: packet distribution over time on one receiver for case one.



(a) Received packets vs. Time. (b) Sent packets vs. Time.

Figure 4.13: Video resolution 800×600: packet distribution over time on one receiver for case one.

packet group. With increased video resolution, the large packets become more centralized around 1000 bytes. We deduce that large-sized packets contain video payload and small-sized packets contain control payload.

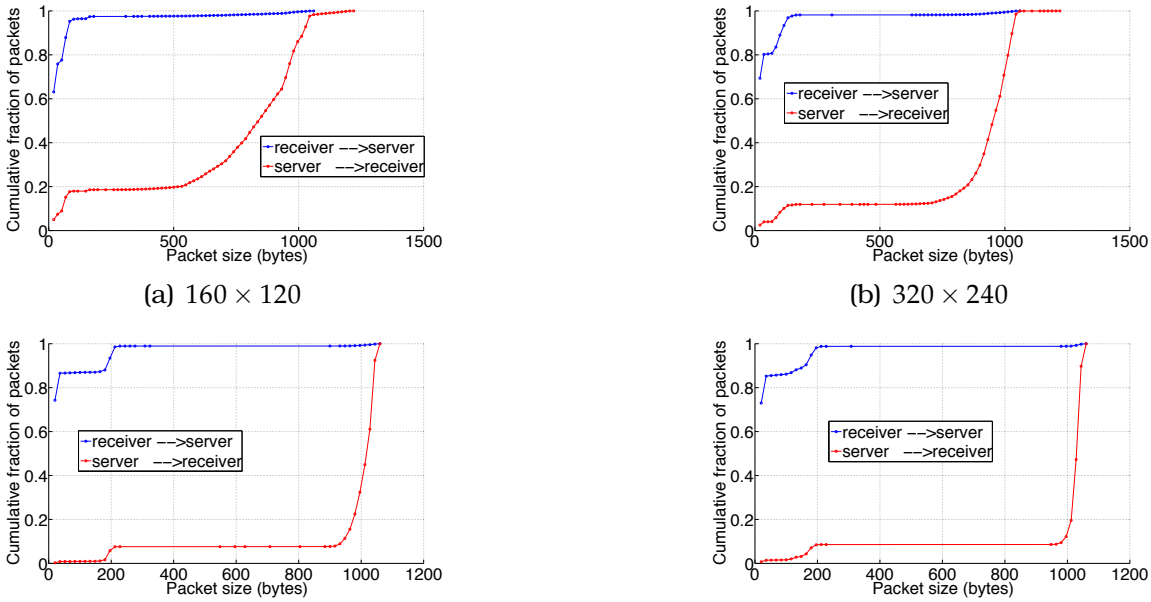
We also find that the receiver acquires large-sized packets in a large delay after session establishment when published video has high resolution such as 640×480 pixels and 800×600 pixels.

In this measurement scenario, only one receiver plays video. It receives video data from FMS. At same time, it also sends video data to FMS. Figure 4.12 presents sent packets from the receiver to FMS. There are large packets with size around 1000 bytes. They apparently are video data. This RTMFP group has two members. The data delivery uses a hybrid pull-push method. As a peer in P2P group, the receiver pushes video data to another peer at certain frequency.

Packet size distribution

This section presents packet size distribution for different published video resolution illustrated in Figure 4.14. The packet size is the size of UDP packet payload in bytes. The cumulative fraction for each packet size is calculated. Among sent packets, the fraction of large packets is quite low. Among received packets, the fraction of large packets is very high and increases with increased

published video resolution. The fractions of large packets for four video resolution are respectively 0.81, 0.88, 0.92 and 0.91. This implies that the ratio of overhead for control message is small for high video resolution.



(a) 160 × 120 (b) 320 × 240

(c) 480 × 360 (d) 800 × 600

Figure 4.14: Packet size distribution on one receiver for case one.

Transmission bandwidth

To estimate video transmission bandwidth in this measurement scenario, we separate packets into two groups according to their size. The payload of the large-sized packet is considered as video data. The estimated video transmission bandwidth for each resolution of published is listed in Table 4.3. They are compared with bandwidth estimated from measurement on publisher's side.

Video Resolution (pixels)	160 × 120	320 × 240	480 × 360	800 × 600
from publisher	183 kbps	345 kbps	930 kbps	1017 kbps
from FMS	220 kbps	324 kbps	1104 kbps	948 kbps

Table 4.3: Estimated video transmission bandwidth for case one..

These numbers indicate the estimated video transmission bandwidth on receiver's side is close to that estimated on publisher's side. The estimated bandwidth on receiver's side is smaller than that estimated on publisher's side in some cases. This can be reasonable when FMS has dropped some video data from the publisher. The video playing on the receiver could have some mosaics. In some cases, the estimated bandwidth on receiver's side is larger than that estimated on publisher's side. This also can be reasonable because RTMFP packet could be padded to meet cipher block size constraints.

4.2.2 Case Two: Two Receivers Within Two Networks

In this measurement scenario, two receivers are run on two computers respectively located in home network and university's wireless network. The publisher is run on a computer in university's fixed network. All three networks are not multicast-enabled. The measurement scenario is illustrated in Figure 4.15. The published resolution is 640×480 pixel. The RTMFP group using fusion multicast has three members: FMS and two receivers. Record all packets on the receiver located in home network using Wireshark 1.6.1. Record also all packets on FMS located in a multicast-enabled network using *tcpdump*.

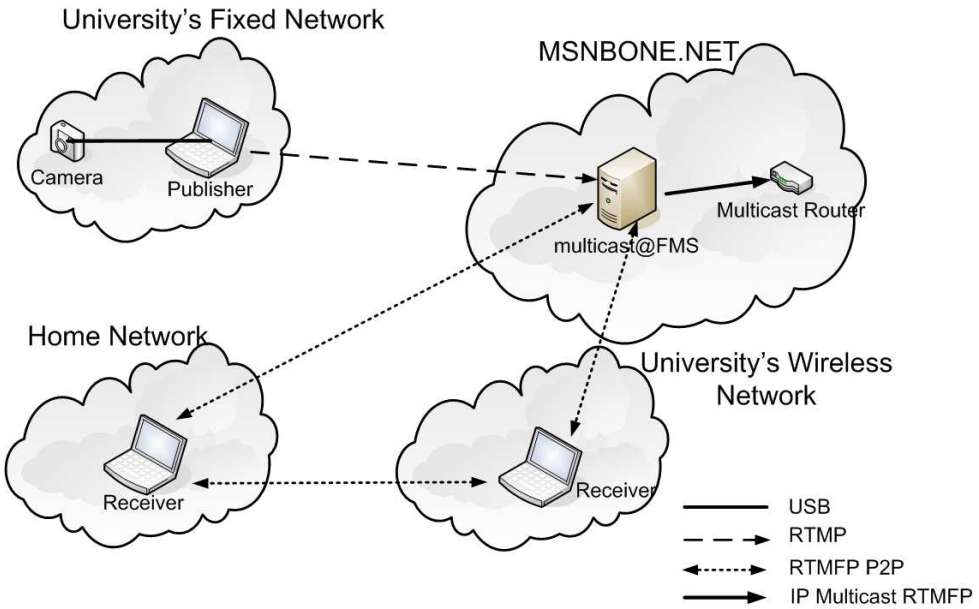
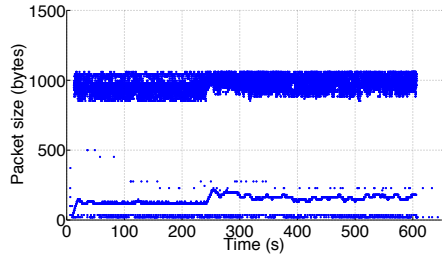


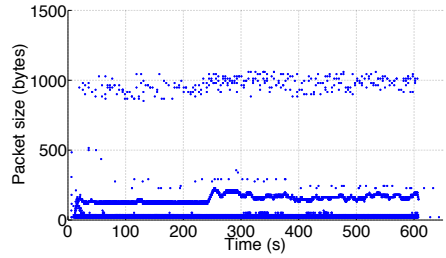
Figure 4.15: Measurement scenario for case two: with two receivers are in two networks. Two receivers and one publisher are running on three computers located on three separate networks.

Packet distribution over time

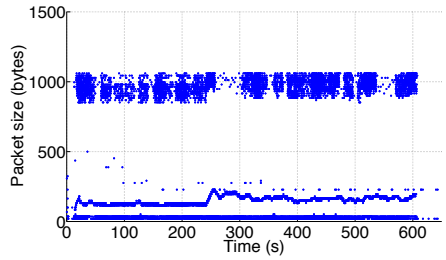
The packet distributions over time are illustrated in Figure 4.16. The packet size is the length of UDP packet payload. The received packets have two sources: FMS and another receiver as illustrated in Figure 4.16(a) and Figure 4.16(c). The receiver also sends packets to FMS and another receiver as illustrated in Figure 4.16(b) and Figure 4.16(d). These packets can be categorized into two groups according to their size: large packet and small packet group. We can consider that the large-sized packets contain payload for video data.



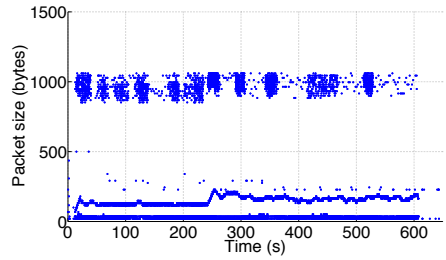
(a) Received packets from FMS.



(b) Sent packets to FMS.



(c) Received packets from another receiver.



(d) Sent packets to another receiver.

Figure 4.16: Packet distribution over time on one receiver for case two.

Packet size distribution

The packet size distribution is illustrated in Figure 4.17. The packet size is the length of UDP packet payload. The cumulative fraction of each packet size is calculated. The percentages of packets with size ≥ 852 bytes are listed in Table 4.4. The receiver acquires more video data from FMS from another receiver. FMS is also video source in this RTMFP group.

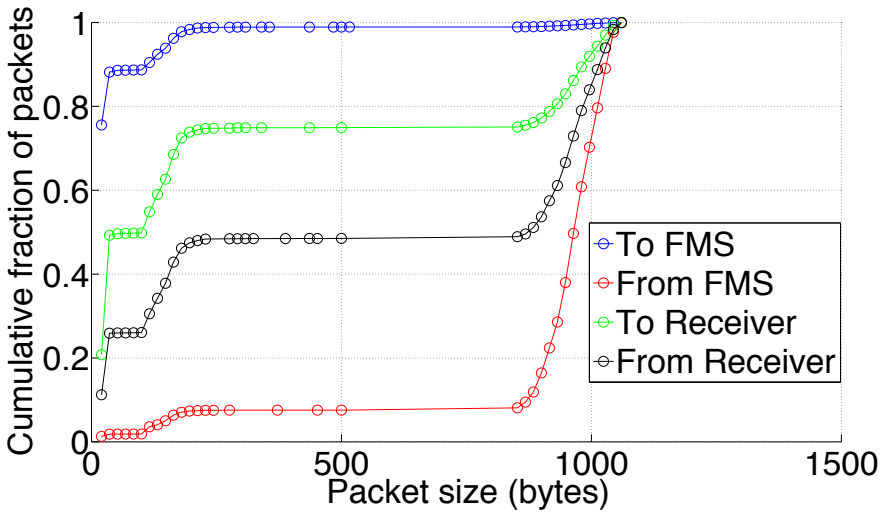


Figure 4.17: Packet size distribution on one receiver for case two.

Packets size ≥ 852 bytes	**Percentage (%)**
to FMS	1.05
from FMS	91.89
to receiver	24.9
from receiver	51.07

Table 4.4: Case two: percentage of packets with size ≥ 852 bytes.

Packets Captured On FMS

In this measurement scenario, the definition of RTMFP group supports both IP multicast and P2P multicast. On receiver's side, there are no IP multicast packets. On server's side, FMS open three ports: two sending to the two receivers and one sending to multicast group. The multicast group is identified using multicast address 224.0.0.254 : 3000. This address is specified on publisher's GUI. The IP multicast packets can only be delivered inside the network where the computer running FMS is located.

Transmission bandwidth

The packets are separated into two group according their size. The packets in large-sized group are considered as their payload contain video data. The estimated transmission bandwidth for receiving and sending video are listed in Table 4.5. In this measurement scenario, the packets on publisher's side are not recorded due to no packet dumping tools available on that computer located on university's fixed network.

	Video Transmission Bandwidth
to FMS	4 kbps
from FMS	629 kbps
to receiver	38 kbps
from receiver	86 kbps

Table 4.5: Case two: estimated video transmission bandwidth.

With the hypothesis that there is no repeated video data on these packets, the total video transmission bandwidth is estimated to be total $629 + 86$ kbps, i.e. 715 kbps. Assume that another receiver also needs bandwidth 715 kbps for its video transmission. It has received video from this receiver with bandwidth 38 kbps. The video transmission bandwidth from FMS is estimated to be 677 kbps. With this assumption, the percent of saved bandwidth on FMS using P2P multicast compared to using unicast is calculated in Equation 4.2.

$$percent_saved\_bandwidth = \frac{38 + 86}{715 + 715} = 8.7\%. \tag{4.2}$$

If there are more than two receivers in the same RTMFP group, this receiver also acquires video data from more than one receivers. More bandwidth on FMS is expected to reduce for streaming with increased number of members.

In fact, the hypothesis about no repeated video data is false due to RTMFP P2P group utilizes also a push data delivery method. This delivery method just delivers video data without checking its existence at first. The video data delivered to FMS can be considered in push. In this measurement scenario, only 4 kbps is used for pushing video data. The bandwidth of pushing video data can be considered to be small.

4.2.3 Case Three: Two Receivers Within Multicast-Enabled Network

In this measurement scenario, two receivers and one publisher are run on three computers located MNSBONE.NET. This network is multicast-enabled. The measurement scenario is illustrated in Figure 4.18. The published video resolution is 160×120 pixel. The RTMFP group using fusion multicast has three members: FMS and two receivers. Record all packets on one receiver using Wireshark 1.6.1.

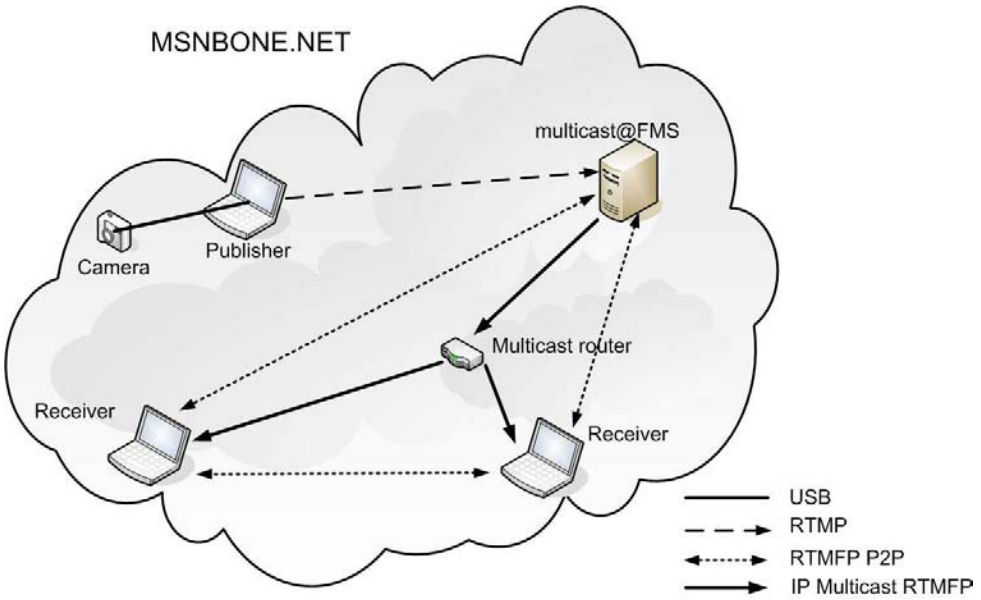
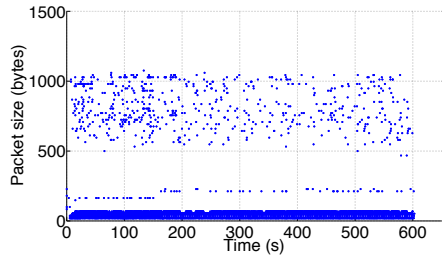


Figure 4.18: Measurement scenario for case three: with two receivers in multicast-enabled network. Two receivers and one publisher are running on three computers located on MNSBONE.NET. This network is multicast-enabled.

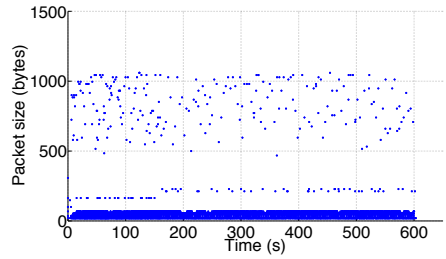
Packet distribution over time

The packet distributions over time are illustrated in Figure 4.19. The packet size is the length of UDP packet payload in bytes. The received packets come from three sources: FMS, another receiver and multicast group as illustrated

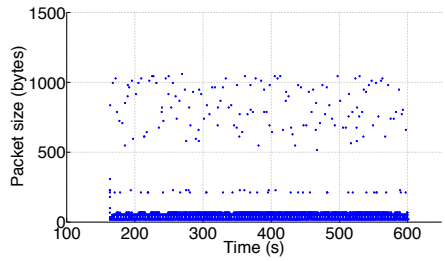
respectively in Figure 4.19(a), Figure 4.19(c) and Figure 4.19(e). The receiver also sends packets to FMS and another receiver as illustrated in Figure 4.19(b) and Figure 4.19(d). The exchanged packets with another receiver start a few minutes later as illustrated in Figure 4.19(c) and Figure 4.19(d). This indicates another receiver joins a few minutes later. The received IP multicast packets have only large-sized packets as illustrated in Figure 4.19(e).



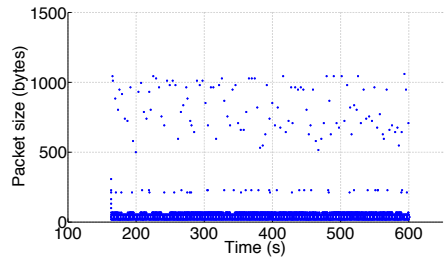
(a) Received packets from FMS.



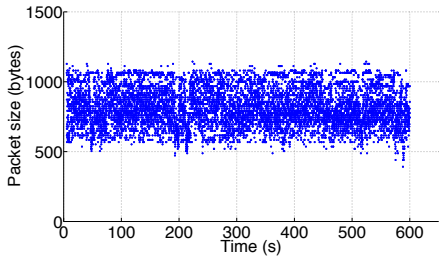
(b) Sent packets to FMS.



(c) Received packets from another receiver.



(d) Sent packets to another receiver.



(e) Received multicast packets.

Figure 4.19: Packet distribution over time on one receiver for case three.

Similar to the case in 4.2.2, FMS and two receivers consist of an RTMFP group. There are exchanged packets among them. The difference is that computers running receivers are located in an multicast-enabled network. The most of received large-sized packets are from IP multicast. P2P multicast and IP multicast are used concurrently, but IP multicast is preferred in this measurement scenario.

Packet size distribution

The packet size distribution is illustrated in Figure 4.20. The packet size is the length of UDP packet payload. The cumulative fraction of each packet size

is calculated. The most of exchanged packets with FMS and another receiver contain payload for control data. The percentages of small-sized packets in this case are listed in Table 4.6. Among IP multicast packets, there are no packets for controlling.

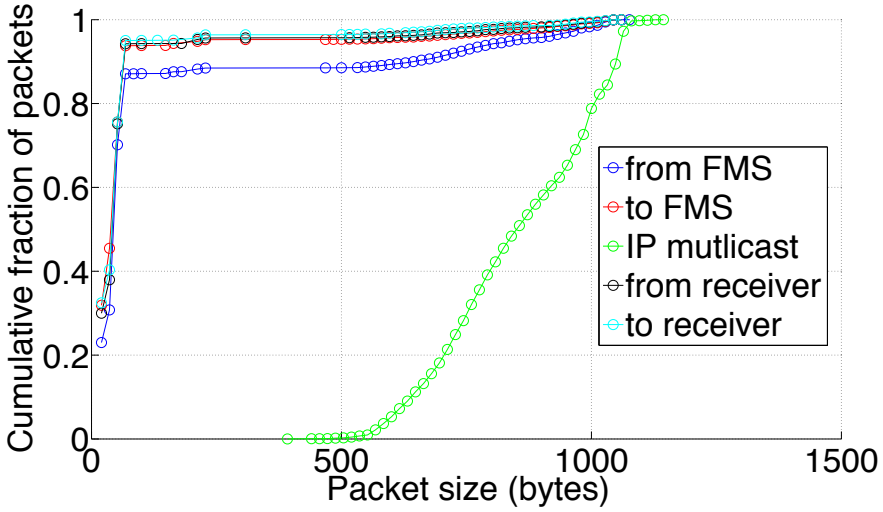


Figure 4.20: Packet size distribution on one receiver for case three.

Packets size < 350 **bytes**	**Percentage (%)**
from FMS	88.46
to FMS	95.23
IP multicast	0
from receiver	95.63
to receiver	96.38

Table 4.6: Case three: percentage of small-sized packets.

Transmission bandwidth

The packets exchanged with FMS and another receiver are separated into two groups. The estimated video transmission bandwidth is calculated using large-sized packets. All IP multicast packets contain video payload. The results are listed in Table 4.7. These numbers indicate that video is delivered mainly in IP multicast.

4.3 Problems in Testing Demonstrator

In order to involve so many scenarios as possible for measurements, we have tried to test the demonstrator in other scenarios. Some measurements have

	Video Transmission Banwidth
from FMS	7 kbps
to FMS	3 kbps
IP multicast	140 kbps
from receiver	2 kbps
to receiver	2 kbps

Table 4.7: Case three: estimated video transmission bandwidth.

similar results, such as a measurement scenario in which two receivers are in home network. Its results are very similar to these in section 4.2.2. Some networks block RTMFP connection or forbid use of uplink bandwidth. The receiver can not work at all on computers located in such networks. There are many available computers in university. But the university's fixed network blocks RTMFP connection. In addition, we have very limited number of available computers. At most three computers have been used in our measurements. Another problem is to test the demonstrator on computers with different geographic locations. The publisher and receiver applications can only be opened in web browsers supporting Flash Player plugin. It is difficult to open web browser on computer remotely for most of operating systems.

Chapter 5

Conclusion

In this chapter, our work is summarized in section 5.1. Section **??** presents some contributions from our work. Some possible work in future is described in section 5.3.

5.1 Summary

In this work, we have implemented a global demonstrator providing efficient live media streaming service using multicast communication. To complete this design, we have first studied different types of multicast communications and their current current implementations in media streaming from the relevant literatures. We figure out that IP multicast is an efficient communication mechanism, but with a lot of limitations in its global implementations. P2P multicast, especially mesh-based, is considered an efficient multicast communication mechanism that can be implemented globally. In addition, mesh-based P2P multicast has been applied in a lot of commercial streaming systems. The business results also show its feasibility and scalability in current Internet. In conclusion, an ideal implementation of such demonstrator shall support both IP multicast and P2P multicast in order to achieve efficiency and globalization at the same time.

Our next step is to investigate some available media streaming systems in current market. Our investigation has focused on Adobe's media streaming system. A new protocol - Real-Time Media Flow Protocol (RTMFP) and a P2P multicast technology - RTMFP P2P group are recently introduced into Flash platform. Adobe claims their new technologies support both IP multicast and P2P multicast. This is consistent with our goal in this work. Flash Player is currently one of most popular media player runtime. We decide to implement the demonstrator on Adobe's Flash platform.

We have first installed and configured Flash Media Server (FMS) on a computer running CentOS server located in MNSBONE.NET. This network is multicast-enabled. FMS is key component of our demonstrator. We also developed two client applications: publisher and receiver. They are HTML pages wrapping

their SWF files. They can be opened in web browser supporting Flash Player plugin. The publisher captures live video from camera and then sends it out. The receiver obtains video and then plays it. The video is associated with an RTMFP group on publisher's side. To access this video, one receiver has to know its RTMFP group. The parameters of group are specified using Multicast Configuration Tool on GUIs of publisher and receiver.

Then we have performed a series of experiments using the demonstrator in different network environments, such as in company, at home and in university. These network environments are in real world without any specified settings. In each experiment, each client application, a publisher or a receiver, is running on one computer. The video captured on publisher's side can be watched on receiver's side with some delay. The packets on network interface are also recorded using packet dumping tools for further analysis. We have investigated the behavior of publisher sending video and RTMFP group management of receivers. We find out that the receivers deliver video data to others in the way P2P multicast works. The receivers prefer IP multicast if they are run on computers located in a multicast-enabled network. We also estimate bandwidth for video transmission and figure out that using multicast can actually reduce bandwidth required on server with increasing number of receivers.

Finally, we figure out several problems that we have met in our experiments. Current demonstrator is only a prototype. It is not global in fact. It is difficult to perform experiments in remote computers. Our experiments are performed with very limited number of computers.

5.2 Contributions

In this work, we have implemented a prototype on Adobe's Flash platform. This prototype utilizes both IP multicast and P2P multicast. It can demonstrate live video streaming in real network environments. We have performed a series of experiments using this prototype. By analyzing measurements in experiments, It is concluded that the use of multicast communication in video streaming can reduce its load on the server. We also conclude that P2P multicast provides a possible solution for global video streaming with increased scalability. IP multicast can be only used inside the multicast-enabled network. The computer installing FMS is also located in this network. Adobe's multicast solution does not overcome its limitations .

5.3 Future Work

This work needs more experiments for further analysis. Firstly, the experiments involving more receivers are needed to further investigate the scalability of this prototype. Our experiments have only involved very limited number of

computers due to available computers in university with no supporting RTMFP connection. RTMFP is the key technology in this prototype. Secondly, the experiments involving more receivers on computers in different geographical location are needed to investigate the global demonstration of this prototype. Our experiments have been performed mostly on computers locally due to difficulties in running client applications on remote computers. One suggestion to involve more receivers is to set up a published video appealing so many watchers as possible in whole world.

Finally, some improvements can be performed in this prototype to make it really globally. In addition to RTMFP, Adobe also supports other streaming protocols such as RTMP. RTMP is TCP-based and not blocked in most network environments. The receiver in the prototype can be modified by shift to an RTMP connection if it fails to establish RTMFP connection. The video streaming based on RTMP is unicast. This modification results in that the prototype can become a real global demonstrator for live video streaming even such modification has decreased the efficiency and scalability of the prototype.

Bibliography

[1] Adobe. Adobe Flash Media Server family. http://www.adobe.com/products/flash-media-server-family.html.

[2] Adobe. Adobe Flash Platform. http://www.adobe.com/platform/whitepapers/platform_overview.pdf.

[3] Adobe. Introducing the Adobe Flash Platform. http://www.adobe.com/devnet/flashplatform/articles/flashplatform_overview.html.

[4] M. Allmand, V. Paxson, and E.Blanton. *TCP Congestion Control*. IEFT, 2009.

[5] Apple. HTTP Live Streaming Overview. https://developer.apple.com/resources/http-streaming/.

[6] S. Banerjee and B. Bhattacharjee. A comparative study of application layer multicast protocols. 2007.

[7] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM*, New York, 2002.

[8] M. Castro, P. Druschei, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: Hight-bandwidth multicast in cooperative environments. *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 37(5):298–313, December, 2003.

[9] Cisco. *Cisco Visual Networking Index: Forecast and Methodology, 2011-2016*. May, 2012.

[10] S. E. Deering. Multicast routing in internetworks and extended lans. *SIGCOMM Proceedings on Communications architectures and protocols*, ACM, 1988.

[11] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Bolensiefen. Deployment issues for the ip multicast service and architecture. *IEEE Network*, 14(1):78–88, 2000.

[12] S. Floyd. *Congestion Control Principles*. IEFT, 2000.

[13] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Transaction On Compuer*, 52(2):139–149, 2003.

[14] G. Grossman and E. Huang. ActionScript 3.0 Overview. `http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html`.

[15] C. Hedrick. *Routing Information Protocol*. IEFT, 1988.

[16] X. Hei, Y. Liu, and K. W. Ross. Iptv over p2p streaming networks: the mesh-pull approach. *Communications Magazine, IEEE*, 46:86–92, 2008.

[17] Y. hua Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an internet broadcast system based on overlay multicast. *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 155–170, 2004.

[18] Y. hua Chu, S. G. Rao, S. Sehan, and H. Zhang. A case for end system multicast. *Proceedings of ACM SIGMETRICS*, 2000.

[19] A. Inc.. *RTMP Specification 1.0*. June, 2009.

[20] A. S. Inc. ActionScript 3.0 Reference for the Adobe Flash Platform. `http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/index.html`.

[21] A. S. Inc. Server-Side ActionScript Language Reference for the Flash Media Server 4.5. `http://help.adobe.com/en_US/flashmediaserver/ssaslr/WS5b3ccc516d4fbf351e63e3d11a11aff5ba-7f8bSSASLR.html`.

[22] J. Kurian and K. Sarac. A survey on the design, applications, and enhancements of application-layer overlay networks. *ACM Computing Surveys*, 43:5:1–5:34, 2010.

[23] M. Kwon and S. Fahmy. Topology-aware overlay networks for group communication. *Proceedings of NOSSDAV*, 2002.

[24] B. Li, S. Xie, G. Y. Keung, J. Liu, I. Stoica, H. Zhang, and X. Zhang. An empirical study of the coolstreaming+ system. *IEEE Journal on Selected Areas in Communications*, 25:1627 – 1639, 2007.

[25] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang. Inside the new coolstreaming: Principles, measurements and performance implications. *Proceedings of the IEEE Conference on Computer Communication (INFOCOM)*, pages 1031 – 1039, 2008.

[26] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, 2008.

[27] N. Magharei and R. Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1415–1423, 2007.

[28] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live p2p streaming approaches. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1424–1432, 2007.

[29] H. page. pplive homepage. `http://www.pplive.com`.

[30] H. page. ppstream homepage. `http://www.ppstream.com`.

[31] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Elimiation trees from overlay multicast. *Peer-to-Peer Systems IV: Lecture Notes in Computer Science*, 3640:127–140, 2005.

[32] L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. *Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III)*, ACM:New York, 2004.

[33] S. Ratnasamy, S. Shenker, and S. McCanne. Towards an evolvable internet architecture. *Proceedings of ACM SIGCOMM*, ACM:New York, 2005.

[34] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, P.Sparks, M. Handley, and E. Schooler. *SIP: Session Initiation Protocol*. IEFT, June, 2002.

[35] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Application*. IEFT, July, 2003.

[36] M. Silverlight. IIS Media Services. `http://www.iis.net/media`.

[37] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE Multimedia*, 18:62–67, 2011.

[38] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. *ACM SIGCOMM Computer Communication Review*, 34:107–120, 2004.

[39] T. Stockhammer. Dynamic adaptive streaming over http - standards and design principles. *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144, 2011.

[40] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31:149–160, 2001.

[41] M. Thornburge. Adobe's Secure Real-Time Media Flow Protocol. `http://tools.ietf.org/html/draft-thornburgh-adobe-rtmfp-04`, Feburary, 2013. Internet Draft for IETF that expires in August 18, 2013.

[42] M. Thornburge. *Advanced P2P with RTMFP: Tips and Tricks*. MAX 2011 Develop, October, 2011.

[43] D. Waitzman, C. Partridge, and S. Deering. *Distance Vector Multicast Routing Protocol*. IEFT, November, 1988.

[44] Wowza. Wowza Media Server. `http://www.wowza.com/`.

[45] B. Zhang, S. Jamin, and L. Zhang. Universal ip multicast delivery. *International Workshop on Networked Group Communication (NGC)*, 2002.

[46] B. Zhang, J. Liu, B. Li, and T.-S. P. Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. *Proceeding of Joint Conference of the IEEE Computer and Communications Societies(INFOCOM)*, 3:2102–2111, 2005.