Adding the RLC and RMDP multicast protocols to the Application Level Active Networking (ALAN) environment

Ibraheem Umaru-Mohammed

January 15, 2000

Abstract

Active Networks permit applications to dynamically load custom programs in order to perform desired operations on user traffic. This innovative approach to networking offers a tractable solution to achieving end-to-end performance improvements, and allows protocols and services to be deployed rapidly. In this paper we introduce the Application Level Active Networking (ALAN) system, and discuss how we can use this system to provide multicast services for distributed applications. We provide a brief technical description of the Receiver-driven Layered Congestion Control (RLC) protocol and the Reliable Multicast data Distribution Protocol (RMDP) that we implemented, and demonstrate the deployment of these protocols within the ALAN active services infrastructure. We conclude by discussing the benefits of Active Networks, and evaluate the acrhitectural strengths and weaknesses of the ALAN system.

Keywords: Active Networks, multicast.

1 Introduction

Active Networks can be observed as a novel approach to networking that aims to permit user-defined computation within the network. This architecture provides applications with a mechanism to load protocol processing code into the network on demand. Traditional networks suffer from problems of slow service evolutiondue to the inflexibility of the existing infrastructure. Active Networks seek to address the problems of integrating new services, and technologies and aim to overcome the problems of poor performance levels and extending the functionality of the network infrastructure.

Clearly, if the user is empowered with ability to inject custom programs into the network, then security is of major concern. The Active Network architecture must provide a framework that allows services to expressed safely, easily and with minimumimpact on other network users. The architecture must be flexibile enough to accomdate new services, and restrictive enough to meet security and performance goals. We believe that the ability to tune the network service to a particular application is an justifiably important issue, if the security properties of network are protected.

Much of the current work in Active Networks can be classified into two main areas [1]: the first approach is known as the *capsule* method, and is based on the concept of having service nodes to observe every packet that arrives at the node as being part of a program. Each one of these packets are interpreted as fragments of a program that contains embedded data. The service nodes evaluate each packet in turn, and perform any necessary computations or modifications to the packet contents. This approach is thought by many to be unrealistic, as it introduces intolerable security and safety issues. The second approach is defined as the *programmable switch* technique, and is based on a system that allows users to place customised programs into service nodes, which can be loaded and executed on demand. The service nodes provide an execution environment for running the customised programs, and also have the capability to perform computations on the traffic passing through the node. The user therefore has the flexibility to send their data through specific service nodes for code processing.

There are a number of existing applications that apply the concepts of active services in traditional networks. This class of services include firewalls (monitor and filter the traffic that comes in or out of a network), web proxies (used to provide a user-transparent service tailored to the serving and caching of web pages), video gateways, and nomadic routers to name a few.

In what follows, we describe the implementation of two multicast protocols, Receiver-driven Layered Congestion Control (RLC) and the Reliable Multicast Distributed Protocol (RMDP), and how they offer a scalable solution to the problems of reliability and congestion control in the global network. We begin by giving a brief introduction to the ALAN system (similar to the programmable switch approach) of active networking before giving a broad introduction to multicast, and providing multicast services on the global network. We conclude by evaluating our work in the context of the architecture of the ALAN system, and discuss the architectural strengths and weakness of the system.

2 The ALAN Infrastructure

2.1 Overview

The ALAN system provides the user with a flexible framework for supporting active services within traditional network boundaries. The programming language used is Java, and the architecture consists of three main components: Dynamic Proxy Servers (DPS), proxylets and dynamic protocol stacks.

The DPS is a service node that provides an execution environment for application level control modules, called proxylets. Proxylets can be dynamically loaded onto the DPS infrastructure, and they usually contain the code that performs the networking and filtering functions. It is possible to place proxylets on regular World-Wide Web [2] (WWW) servers, and to reference them by a Universal Resource Locator (URL). They may also be obtained from protocol servers. We discuss the workings of these components in turn, in the following sections.

2.2 The Dynamic Proxy Server

Dynamic Proxy Servers (DPS) can be defined as nodes deployed throughout the network to perform special servicing functions. The DPS can await a set of requests (communicated via RPC or RMI) to load and execute proxylets on behalf of a third party. The set of requests that can be communicated to the DPS are defined below.

Load: The *load* request is used to pass references for proxylets to the DPS, in order for the DPS to perform a number of acceptance checks on the referenced proxylet. This includes validation, loading, permission and security checks. If the proxylet is acceptable, then it is loaded onto the DPS and awaits the *run* request. [1]

Run: Once the proxylet has been loaded successfully, the server awaits a call to the RUN method, which provides the proxylet with a set of arguments. The arguments are used to specify hostnames, particular protocols or transcoders to be used or further downloaded.

Modify: This request allows the state of a running proxylet to be changed.

Stop: This request allows the server to force termination of a running proxylet.

The DPS is central to the workings of the ALAN system, and it is envisaged that they will be deployed all over the global network. In a typical end-to-end communication path, one or more DPSs will be used as service points to enhance the performance and to provide services tailored to a specific application. Currently, the locations of our DPSs are known beforehand, as there are no dynamic DPS resolutions algorithms currently deployed. A mechanism is required

[1] As proxylets are loaded by URL reference, it is possible for the server to use a local WWW cache for references that have already been loaded.

to discover best placed DPSs across an end-to-end path distributed across the network. An approach similar to routing protocols is presently being developed that allows DPSs to be discovered dynamically.

2.3 Proxylets

Proxylets are control modules that contain the code for processing the necessary filtering and networking functions of a service. It is possible for proxylets to be self-contained units. However, the downloading of parts of required protocol stacks (dynamic protocol stacks) from Dynamic Protocol Servers encourages code sharing and reuse. The proxylets are executed on DPS machines, and highlights an important mechanism in the system for providing new services and enhancing end-to-end performance and utilisation.

The proxylet interface is implemented in a similar fashion to the way interfaces are implemented on applets running a WWW browser. The DPS machine executes the proxylets by invoking the methods of the proxylet interface: `init()`, `run()`, `modify()`, and `stop()`. Once a proxylet has been downloaded successfully, it is stored as a collection of class files bundled together in the form of a single Java archive (Jar [2] file). Each proxylet exists as one Jar file that is subsequently placed on a WWW server, (or protocol server) so that it can be referenced via an URL. When The DPS receives a *'load'* call, it initialises the proxylet to be loaded by calling the `init()` method of the proxylet.

The sequence of events are described below:

1. The DPS loads the proxylet (Jar file) from a WWW server via an URL. The class files of the Jar file, contain the machine independent byte codes for the requested proxylet, which are loaded by the Dynamic Proxy Server.

2. The DPS instantiates an instance of the proxylet object, and then calls the init() method of the proxylet class to initialise the downloaded proxylet.

3. The server invokes the proxylets' `run()` method, which starts the execution of the proxylet. The `run()` method is where most of the code for networking and filtering is encapsulated and it is possible to supply arguments to this request.

4. When the server no longer needs the proxylet, it invokes the proxylets `stop()` method. It may be neccessary to force termination of a proxylet which may refuse to terminate naturally.

2.4 Dynamic Protocol Stacks

Communication systems require matching protocol stacks to enable entities to communicate and interoperate. In the ALAN system, Dynamic Protocol Stacks have been designed to simplify the interoperability process, by allowing code for

[2] The Jar (Java archive) file is a collection of class files produced as a result of the Java compilation process.

Dynamic protocol stacks to be written once and placed on repositories where they can be shared. These protocol stacks can then be downloaded onto either the end systems or the dynamic Proxy Servers, so that end systems (or proxylets) can adopt the same stack.

2.5 Control Architecture

The ALAN system provides a Control Interface for making explicit connections to a particular DPS. This interface uses Remote Method Invocation (RMI) to communicate with the remote DPS that we wish to run the proxylet on. It is also envisaged that specialised control interfaces will be developed to enable user interaction. to achieve this, the Monitor Interface has been developed, that allows the user to observe the execution of proxylets on a given DPS machine. The Monitor Interface will enable the user to take control over the running of proxylets on a DPS machine.

2.6 Security

Without suitable safeguards and security measures, the active network concept will remain unacceptably fragile. The security issues of the ALAN system are continually being addressed. However, the application layer solution propose by the ALAN system uses a control protocol that makes the situation slightly more tractable. It is quite clear that stringent restrictions have to be placed on the user, as they have the flexibility to execute arbitrary code on foreign machines. The ALAN system has defined the following security rules in order to make the environment safe:

- Only trusted hosts are allowed to submit requests to a DPS
- DPSs can't download proxylets from untrusted WWW servers.
- Verification checks are performed on proxylets to ensure that they are acceptable.
- DPS can only communicate with proxylets via the interface provided, thereby protecting the DPS from modifications.

The Java programming language itself also adds useful security features. The ALAN system uses one such mechanism that allows one to add their own security manager to the run-time system. This mechanism has been used to prevent proxylets from reading or modifying local files on the DPS machine. Ongoing work in this area aims to disallow certain local operations, and to relax the constraints on the originating source of the proxylet.

3 Multicasting

Traditional network computing applications involve communication between two computers. They provide a *point-to-point*, unicast service with two well-defined connection endpoints. Unicast communications are used to serve many, if not most of todays communication needs. However, there are many important applications which require another model for communication. A large number of emerging applications require a model that encompasses simultaneous communication betweeen groups of computers. These applications require a *point-to-multipoint* service of communication.

When a computer wishes to send data to many receivers simultaneously, it may follow one of two options [3]: it may repeatedly transmit the data to each individual receiver or use a technique called *broadcast*. *Repeated transmissions* suffer from scaling difficulties for large groups, and can be costly in terms of the network bandwidth required [3]. The broadcast approach allows applications to send one copy of data to a broadcast address. This method allows the network to stop broadcasts at network boundaries or to send the message everywhere. Clearly,it can be seen that the latter option signifies a significant usage of network resources, especially when the size of the group of receivers is small.

The Internet offers an alternative solution to the point-to-multipoint problem. *IP Multicasting* provides the Internet with an attractive approach that saves precious bandwidth for point-to-multipoint data distribution. Multicasting allows applications to send one copy of each packet, to an address that represents a group of computers that wish to receive the data. In this way, data can be distributed to the hosts that have explicitly expressed an interest in receiving the data for that multicast group. This vision of point-to-multipoint communication relies on the network to forward packets to the relevant networks that wish to receive them, and to stop any others from receiving the data.

The *MBone* [18] (multicast backbone) is the multicast experimental infrastructure that has been developed to support network level multicast. The forwarding of multicast data in the Mbone is implemented with hosts using a group membership protocol known as the Internet Group Management Protocol (IGMP) [4]. Unlike unicast addresses for point-to-point communication, hosts can dynamically subscribe to multicast addresses and by doing so cause multicast traffic to be delivered to them. Multicast data is unlike unicast data in that the sending entity for multicast has to send the traffic to a special multicast group address. The receiver may then receive this data by using IGMP to tell its local router that it wishes to receive packets addressed for the multicast group address. This is called the *join* phase, and once a receiver has joined a group, the router enables the packets to be forwarded to that receiver. Likewise, receivers may choose to leave a group, and do so by informing the local router that they are no longer interested in receiving data from the group address. This is called the *leave* phase, and the router proceeds to cease the delivery of packets to that receiver.

In order to support the demands of emerging multicast applications, it is important for us to build networks which can rapidly adapt to these requirements and provide efficient multicast services. Active Network technology is one such approach that allow us to enhance the performance and utilisation problems

[3] as the same information has to be distributed multiple times

encountered by multicast applications. For example, today's MBone is unable to accomodate and satisfy the diverged requirements of a set of multicast receivers. However, Active Networks permit the application to achieve optimal data rates, by specifying the network to take into account factors such as data rates, data formats, network access speed, and end-system performance.

4 Distributed Multicast Demonstrations

Because of the difficulties associated with scalable multicast protocols, they are only a few working implementations of scalable multicast solutions to the problems of reliability and congestion control. However, none of these implementations have been accepted universally. In this section, we introduce two multicast protocols that offer solutions to these problems that we developed with the ALAN proxylet infrastructure. The first protocol is a useful congestion avoidance protocol for multicast applications, and is called the Receiver-driven Layered Congestion Control Protocol (RLC). Our second protocol, Reliable Multicast data Distribution Protocol (RMDP), is a reliable multicast transfer protocol that is used in conjunction with RLC to achieve TCP-like control congestion control. Both protocols were developed by L.Rizzo and L.Vicisano [5, 6].

4.1 RLC

The Receiver-driven Layered Congestion-contol (RLC) algorithm provides us with a useful building block for providing point-to-multipoint congestio control for multicast applications. The protocol is based on a receiver-driven layered congestion-control algorithm which is TCP-friendly and suitable for data transfer over an IP multicast service such as MBone. It uses a layered organisation of data to support variable transmission rates, and provides the receivers with the flexibility to join one or more multicast groups, in order to adapt to the available network bandwidth. This gives the receivers the flexibility to make decisions autonomously.

RLC uses intelligent techniques for synchronisation, to control congestion levels behind shared bottlenecks. These techniques work by ensuring effective cooperation between receivers that share a given bottleneck. Full-scalability is achieved by keeping the sender side free of any feedback mechanisms, and by ensuring the receivers don't use explicit forms of group membership.

4.2 RMDP

RMDP is a scalable multicast protocol that supports the reliable distribution of data to a set of receivers. The version of RMDP defined in this paper is used in conjunction with the RLC multicast congestion control protocol defined above. RMDP aims to overcome the difficulties in ensuring reliable data delivery to large groups and adaptability of heterogenous clients in reliable data transfers.

RMDP achieves reliability by using Forward Error Correction (FEC) techniques, [4] that is inherently tolerant to packet losses. The protocol controls congestion by adopting an efficient TCP-like congestion avoidance alogorithm to accomodate individual multicast receiver rates.

5 Implementation

The integration of the RLC and the RMDP protocols with the active network, has been done using similar implementations. The desire was to provide a transparent service, avoiding any radical changes to the natural workings of web servers and browsers. The reality of security have placed a profound effect on the design and interaction of proxylets with DPS. For reasonable security reasons, common support for servlets was discouraged. One consequence of this limitation was that the server side *had* to be deployed as a proxylet. One possible solution would be to deploy the client side as a plugin (or applet). However, a better approach seemed entirely reasonable, if both the client and the server sides were both implemented as proxylets. In this way, we could then have the users configure their browsers to point at the appropriate proxy server, a local DPS machine.

The process of development used in this implementation proceeds in three phase. The first phase was concerned with the way the objects were to communicate, and where there would be located. From this, we deduced reasonable interfaces, and methods of communication, which enabled us to design the component proxylets. This included modifying existing code for RLC and RMDP, and implementing, deploying and testing them as proxylets.

The second phase was to further decide on exact locations of the objects and communication methods. We decided on the arguments, and At this stage, it was necessary to develop tools for analysis and testing. We used a simple program to inspect the data being transmitted to a multicast group address, and developed a simple applet to represent an application requiring multicast data.

The final phase was to integrate the proxylets with the web transparently. This involved determining a suitable method of communication between the *webcache* proxylet and the RLC/RMDP proxylets. The webcache proxylet was key to the vision of transparency to both server and browser, and is discussed further in the section below.

5.1 The Webcache Proxylet and Other ALAN experiments

The webcache proxylet provides us with a scheme that allows us to be transparently in the path of HTTP requests. This is achieved by configuring the proxy variable on the user's browser to the DPS machine that is running the webcache proxyler. This makes it possible for the webcache proxylet to observe the content types of the browser requests, and compare them with its table of

[4] a redundant data encoding mechanism.

mime content types on which it is able to perform special operations. The webcache proxylet may then add *value* to the content types that it has stored in its table entries. Otherwise, it simple relays the request to the real local WWW cache machine. For example, if we wanted to perform special processing on "audio/basic" content types, we would store this as an entry in the webcaache proxylet table of mime content types. Then, when the webcache proxylet observes that the content type being returned is of this type, it is in a position to identify a nearby DPS machine. In the current version of the webcache proxylet, there is prior knowledge of the location of DPS. However, there is ongoing work in this area, in able to support dynamic DPS location discovery.

A number of other experiments have been performed that provide different services to applications. A few of these are described below.

The Audio Transcoder Proxylet: This proxylet has been used in conjuction with the webcache proxylet to provide a transparent WWW audio streaming service. The audio transcoder uses HTTP/TCP to download audio samples from WWW servers and then converts the incoming audio stream to a RTP stream that is transmits to the users machine. The webcache proxylet is used to send a URL redirect to the users browser in order to load a WWW page that contains a suitable audio tool configured as an applet. This mechanism makes it possible to deploy an audio player applet without special configuration of the WWW browser.

The Compressor/Decompressor Proxylets: These proxylets are used to perform special processing to files in order to enhance the performance of HTTP connections. The webcache proxylet is used to load a decompression proxylet on a local DPS machine, and a compression proxylet on the remote machine. The compression and decompression of the files can then be performed transparent to the WWW browser and server.

The Multicast Reflector Proxylet: This proxylet was developed to provide a MBone connection to hosts in Australia. At the time the proxylet was written, Australia wasn't connected to the global MBone. The reflector proxylet provides a mechanism for isolated networks to join multicast sessions. To demonstrate this, a reflector proxylet was executed at a machine at the Information Sciences Institute (ISI) located at the Massachusettes Institute of Technology (MIT). This reflector listens on a session description channel and forwards packets unicast to a DPS machine at the University of Technology, Sydney (UTS) running a peer reflector proxylet that multicasts the packets locally. It was thus possible to tunnel session information from ISI to UTS, via the reflector proxylet.

The Tcpbridge Proxylet: The Tcpbridge proxylet was developed to provide dedicated TCP connections to segments of the network that experienced poor latency. An observation was made that the direct network path from Southern France to UTS in Australia can be very poor. Far better response times could be gained b y first logging into a machine at University College

London (UCL) and then logging into UTS. The proxylet was used to improve the latency observed for such a connection. The workings of the proxylet are fairly simple and straightforward. The proxyet accepts connections on a certain port and then in turn makes a connection to another host and port, with the data for both directions being relayed via the bridge (cf. snoop solution to wireless tails).

5.2 Proxylet Design

In this section, we aim to describe the workings of the sender and receiver side proxylets for RMDP. The version of RMDP used incorporates the functionality of RLC. We defined a simplified model of communication which attempts to achieve an effective method of communication between the objects.

We chose to implement seperate proxylets for the client and the server sides of communication, to achieve a simple and effective approach that avoids any unnecessary modifications to web servers and web browsers.

A diagram demonstrating the proposed model of communication is shown in Figure 1.

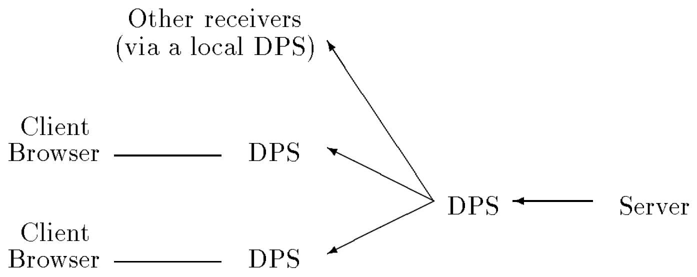

Figure 1: Proposed ALAN model of multicast communication.

The model above aims to provide a WWW transparent multicast service using the ALAN proxylet system. It was envisaged to have each DPS running a proxylet for a particular operation. For example, the DPS near the server would run the sender proxylet, and the other DPSs (for the clients) would be running the receiver side proxylets. These proxylets are discussed further in the following sections.

5.2.1 The Sender Side Proxylet

The sender side proxylet can be defined as the *point* of the *"point-to-multipoint"* method of multicast communication. Our implementation was fairly simple, and represents a transmitter that sends packets to a group multicast address, as soon as it receives a request from the first receiver. This transmission is continued until there are no more active receivers. This is unlike the continous stream of data approach used by RLC [5].

Our vision for component interaction was kept simple, and is shown in Figure 2. The sender side proxylet needs to be provided with the necessary arguments (number of layers, base port/multicast address, time-to-live field and inter-packet time) for proxylet that performs the particular operation. This is achieved with the aid of the webcache proxylet. The webcache proxylet will in the future load proxylets onto strategically located DPS machines. However, in order to make use of the webcache proxylet transparency mechanism, the user first needs to configure the proxy variable of its browser, to be directed at a local DPS machine running the webcache proxylet. This allows the webcache proxylet to observe all HTTP requests made by the browser transparently.

When the user makes a request to a WWW server, the server responds with the requested page preceded by a mime content type. The webcache proxylet has a table of mime content types of which it may perform special operations on. When the webcache proxylet identifies the content type for performing RLC/RMDP sender side operations, it responds by loading the sender side proxylet on a suitable DPS machine near the WWW server. Ongoing work is being done in this area, and as of yet this dynamic component has not been integrated with the webcache proxylet. In the current version, the location of DPS machines are known beforehand.

However, once an appropriate DPS has been located by the webcache proxylet, it loads the sender side proxylet onto the identified DPS machine. If this is succesful, the sender side proxylet awaits a TCP connection from the remote browser. This is made possible by the intermediary webcache proxylet, as it is able to send a URL redirect to the browser, in order to load the necessary applet representing the application. One consequence of this approach is the restriction on applets, that they can only open connections to the host from which they were downloaded from. This imposes the restriction that the applet has to be downloaded from the "dynamically discovered" DPS machine which runs the sender side proxylet. However, this approach provides us with a mechanism for passing arguments and recieving data packets from the DPS machine. When the TCP connection arrives from the user's browser, the DPS machine running the sender side proxylet is able to pass the transmitted packets to the application if required.

[5] This is because an effort is made to achieve maximum quality for the available bandwidth

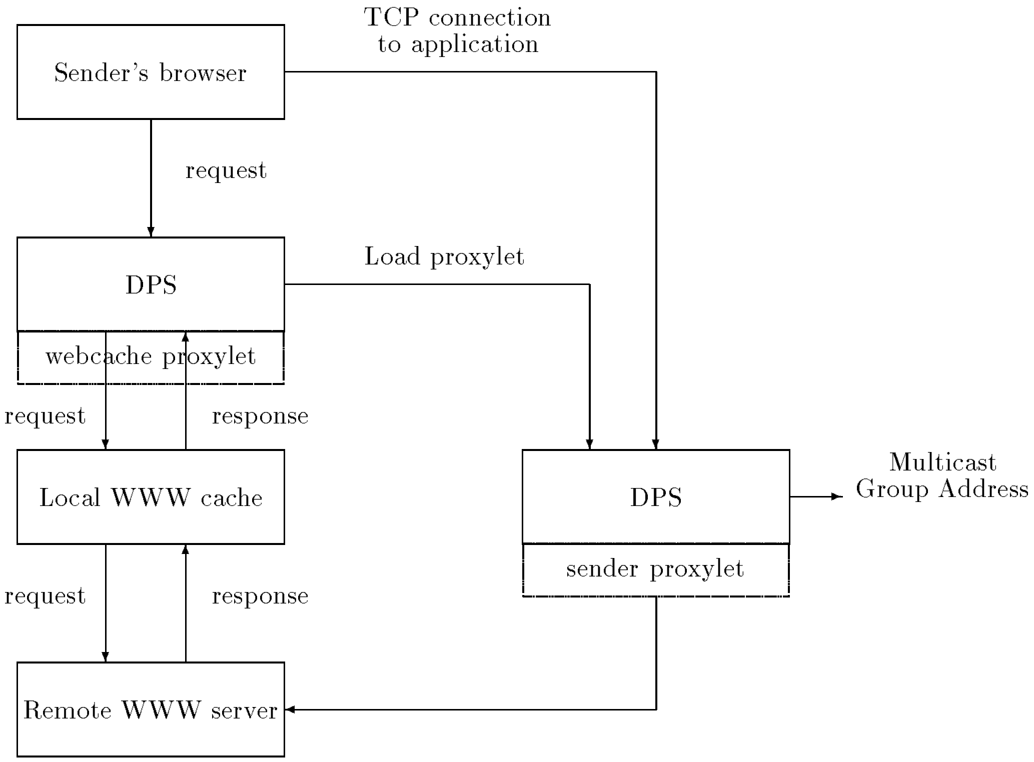

Figure 2: Object interaction for sender side proxylet.

5.2.2 The Receiver Side Proxylet

The components of the receiver side proxylet interoperate in a similar fashion to that of the sender side proxylet. Here, when the receiver makes a HTTP request for a WWW page, the webcache proxylet identifies the mime content type returned by the WWW server, and proceeds to load the receiver side proxylet on a DPS machine close to the receiver's browser. The webcache proxylet also sends a URL redirect to the browser to load the page that contains the application. In our case, we have developed a simple applet, that makes a TCP connection to the local DPS machine running the receiver side proxylet. The DPS machine awaits this connection on a specified port, and responds by streaming the multicast data packets to the user's browser. Figure 3 shows the interaction between the various components.

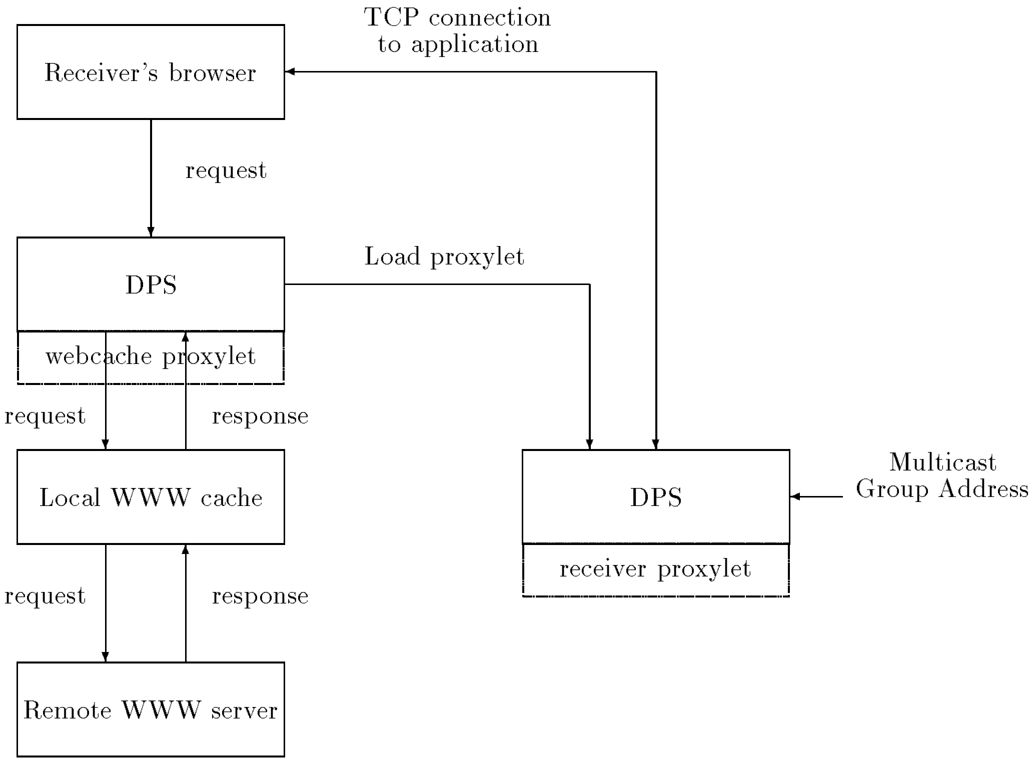

Figure 3: Object interaction for receiver side proxylet.

6 Evaluation

We have successfully implemented the RLC and RMDP scalable multicast protocols with the ALAN system. The protocols were written in Java, which is the language used in the ALAN network environment. One of the advantages of using this system was the ease at which applications could be written, deployed and executed within this environment. Another advantage of this system is the period of deployment. The deployment time of the proxylets was very short, and simply required the compilation of the package of classes into a Jar file, and placing it on a WWW server.

We chose to implement seperate proxylets for the client and the server sides of communication, as this represents a simple and effective approach that avoids any unnecessary modifications to web servers and web browsers.

7 Summary

We have introduced the Application Level Active Networking (ALAN) architecture, and have discussed how this system is used to perform dynamic user-defined computation within the network. We have given a brief technical description of the RMDP and RLC multicast protocols, and how they have been integrated with the ALAN system. We believe that these services extend the functionality of ALAN, and can be used as a testbed for studying issues such as scalability, reliability and congestion control for multicast services.

We have developed a simple demonstration of a distributed file-transfer tool for multicast applications using the ALAN architecture. We found ALAN's use of Java bytecodes and Java virtual machines relatively straightforward and made proxylet deployment and execution relatively simple and efficient to operate. Our work has addressed some of the important opportunities that active services provide for end-to-end performance improvements. We were also able to experience some of the benefits that active networks propose, such as rapid deployment time and flexibility with respect to supporting new services and applications.

In the future, we believe that networks will become increasingly flexible, and that software-intensive approaches such as ALAN offer important opportunities to realising this goal. Although we are still only in the early days of active networking exploration, the ALAN system demonstrates a realistic approach to rapidly adaptable networks and future active service development and deployment.

8 Acknowledgements

The ALAN work and the RLC/RMDP additions was funded by British Telecom Laboratories under the HIPPARCH (High Performance Protocol Architecture) and ALPINE (Application Level Programmable Inter-Network Environment) projects. The ALPINE project stems from interest in programmable networks for telecommunications. HIPPARCH aims to investigate novel architectures for high performance communication protocols that implement the principles of Integrated Layer Processing (ILP) and Application Layer Framing (ALF).

Our thanks go to Lorenzo Vicisano and Luigi Rizzo for developing the RLC and RMDP protocols. We thank Prof. Micheal Fry and Atanu Ghosh for their work on ALAN and developing DPS software.

Finally, I would like to save my largest thanks to Prof. Jon Crowcroft, for the generous support and kind assistance provided throughout this paper. Thank you also for the helpful comments, notes and technical guidance recieved.

References

[1] D.Tennenhouse, J.Smith, W.Sincoskie, D.Wetherall, G.Minden, *"A Survey of Active Network Research", IEEE Communications Magazine, January '97.*

[2] T.Berners-Lee, *"The World-Wide Web initiative". In Proceedings of the International Networking Conference (INET), August '93*

[3] M.Handley and J.Crowcroft *"Internet Multicast Today", The Internet Protocol Journal*

[4] W.Fenner, *"Internet Group Management Protocol, Version 2", Internet Draft, January '97.*

[5] L.Vicisano *"RMDP: Reliable Multicast data Distribution Protocol", http://www.cs.ucl.ac.uk/external/L.Vicisano/rmdp.*

[6] L.Vicisano *"Receiver-driven Layered Congestion control", http://www.cs.ucl.ac.uk/external/L.Vicisano/rlc.*

[7] L.Rizzo and L.Vicisano, *"A Reliable Multicast data Distribution Protocol based on software FEC techniques", The Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communications Systems (HPCS '97), June '97*

[8] L.Vicisano, *"Notes on a cumulative layered organisation of data packets across multiple streams with variable-rate", http://www.cs.ucl.ac.uk/external/L.Vicisano/layers.ps .*

[9] L.Vicisano, L.Rizzo and J.Crowcroft *"TCP-like congestion control for layered multicast data transfer", INFOCOMM '98 paper.*

[10] L.Vicisano and J.Crowcroft *"One to Many Reliable Bulk-Data Transfer in the MBone", The Third International Workshop on High Performance Protocol Architectures HIPPARCH '97, June 97*

[11] L.Rizzo and L.Vicisano *"RMDP: an FEC-based Reliable Multicast protocol for wireless environments", Mobile Computing and Comunications Review, Volume 2, Number 2*

[12] M.Fry and A.Ghosh, *"Application Level Active Netwroking", Fourth International Workshop on High Performance Protocol Architectures (HIPPARCH '98), June 98*

[13] *"The ALAN Programming Manual", http://dmir.socs.uts.edu.au/projects/alan/prog.html.*

[14] D.Tennenhouse and D.Wetherall, *"Towards an active network architecture", Computer Communication Review, April 96.*

[15] E.Amir, S.McCanne and R.Katz, *"An active service framework and its application to real-time multimedia transcoding", ACM SIGCOMM Computer Communication Review, Oct 98.*

[16] A.Banchs, W.Effelsberg, C.Tschudin and V.Turau *"Multicasting Multimedia Streams with Active Networks", International Computer Science Institute, Technical Report, '97*

[17] S.McCanne, V.Jacobson and M.Vetterli, *"Receiver-driven Layered Multicast", SIGCOMM '96, August 96.*

[18] V.Kumar, *"MBone: Interactive Multimedia On The Internet.", Macmillan Publishing 1995.*

[19] J.Crowcroft, M.Handley and I. Wakeman *"Internetworking Multimedia", UCL Press.*