

Active Information Networks and XML

Ian Marshall¹, Mike Fry², Luis Velasco¹, Atanu Ghosh²

¹BT Labs, Martlesham Heath, Ipswich, IP5 3RE, Ian.W.Marshall@bt.com,

²UTS, Sydney, NSW2007, Australia, atanu,mike@socs.uts.edu.au

ABSTRACT

Future requirements for a broadband multimedia network are discussed and a vision of the future network is presented. Three key needs are identified; rapid introduction of new services, dynamic customisation of services by clients, and minimal management overhead. Application layer active networking, perhaps the most pragmatic and immediately realisable active network proposal, is a potential solution to all three. Combining eXtensible Markup Language and Application Layer Active Networking yields strong benefits for networked services. A Wide range of applications can be developed based on the flexibility of XML and the richness of expression afforded by the metadata. A system of network intermediaries based on caches, which are also active and driven by XML metadata statements, is described.

KEYWORDS

Active network, Cache, Active Node, Proxylet, XML, HTTP.

INTRODUCTION

The characteristics and behaviour of future network traffic will be different from the traffic observed today, generating new requirements for network operators. Voice traffic will become another form of data, most users will be mobile, the amount of traffic generated by machines will exceed that produced by humans, and the data traffic will be dominated by multimedia content. In the medium term the predominant multimedia network application will probably be based around electronic commerce capabilities. Operators will therefore need to provide a low cost service, which offers an advanced global trading environment for buyers and sellers of any commodity. The e-trading environment will be equipped with all the instruments to support the provision of a trusted trading space. Most important is the ability to support secure transactions over both fixed and mobile networks. Networks will thus need to be robust, contain built in security features and sufficiently flexible to address rapidly evolving demands as other unforeseen applications become predominant.

Existing networks are very expensive, and the deployment of new communication services is currently restricted by slow standardisation, the difficulties of integrating systems based on new technology with existing systems, and the overall system complexity. The biggest cost is management. The network of the future will need to be kept as simple as possible by using as few elements as possible, removing duplication of management overheads, minimising signalling, and moving towards a hands off network.

The simplest (and cheapest) current networks are multiservice networks based on powerful ATM or IP switches. New transport networks are designed on the basis that nearly all applications will eventually use internet-like connectionless protocols. The difficulties of adding services such as multicast and QoS to the current internet demonstrate that even these simpler IP based networks will require additional mechanisms to enhance service flexibility. The simple transport network will thus need a flexible service surround. The service surround will provide a trusted environment, with security features (for users, applications and hosts), QoS support, application specific routing, automatic registration and upgrade for devices connected to the transport network. It will also enable Network Computing facilities such as secure gateways, application layer routers, cache/storage facilities, transcoders, transaction monitors and message queues, directories, profile and policy handlers. Such a service surround will likely be based on some form of distributed middleware, enabling the features to be modular and interoperable. The service surround must enable rapid introduction of new features by the operator. In order to minimise the management overhead clients will directly control which features should be used for a particular session, without operator intervention. The network will thus need to know nothing of the semantics of the session. To achieve this a middleware based on some form of active services or active networks will be required.

Active networks (Tennenhouse)

Active networking was originally [TENN] a proposal, by Tennenhouse at MIT, to increase network flexibility by adding programmes, that are intended to run on network

devices that the packet encounters, to the packet header. This is referred to as the capsule approach. There are a number of problems;

1. The maximum transport unit (MTU) size in the internet is typically 565 bytes. This will likely be upgraded to 1500 bytes in the near future, however it is clear that if there is to be a programme embedded in every packet the programmes must be very small, even if the programme is not confined to the header. This severely restricts the flexibility that can be offered, although it has been shown that copy instructions to emulate multicast can be embedded in packet headers in some circumstances.
2. It has been proposed that only those packets initiating flows should carry programmes. However, it is common for the packets in an individual flow (such as a document retrieval) to use multiple routes across the internet. This is a result of the routers having the freedom to use the best *available* route at any time, so as to maximise network resiliency. Therefore, in order for a programme to be applied to all packets in a flow, either the route for all subsequent packets in the flow must be pinned, so that all packets flow through the node where the programme was loaded, or, the programme must be copied to all nodes on valid routes. The second option is clearly impractical. The first option is currently not possible, and in any case creates an undesirable reduction in network resilience.
3. The proposal envisages programmes being supplied by network clients. However service operators will never permit third party programmes to run on their equipment without a strong guarantee that the programme will not degrade performance for other users. Such a guarantee requires the programmes to be written in a language in which behaviour is verifiable through pre-run checks, resource usage can be tightly controlled and termination is guaranteed. The Safetynet [WAK] project at Sussex University in the UK is designing a promising language, but the research is still at a very early stage. Since it will be extremely hard to create interesting programmes in a language which is simple enough to enable resource control and termination guarantees, the flexibility offered by this approach is probably somewhat limited, even when the language is mature.
4. The programmes are intended to be added to the switch control kernel in the router. All the known approaches to making the kernel extensible degrade performance. Packets which do not require router programming will thus suffer an unacceptable performance penalty.
5. Undesirable interactions between programmes and network features are almost impossible to predict and control. For example a mobile client will potentially send programmes to several routers where they are used once, then do not receive the acknowledgement packets that would terminate the programmes as the acks are routed to the clients current location.
6. Standards for the interface offered by active routers must be developed before any service based on this proposal could be offered. Appropriate standards are not even being discussed at present.

Despite the manifest difficulties inherent in this proposal it has succeeded in highlighting an important requirement, and stimulating discussion amongst a previously disparate community of researchers attempting to develop more immediately realisable means to resolve the requirement. The main threads are summarised in the next section.

Active networks and services

The first response to Tennenhouse was a somewhat different flavour of active networking, in which the packets do not carry programmes but transport layer header flags indicating the desirability of running a programme [ALEX97]. This approach attempts to resolve the issue of restricted programme size, and potentially gives network operators the freedom to choose an appropriate programme of their own which has been tested. However, the proposal makes no progress on the last 3 issues, and the range of flags could not be large as the space available in the transport layer header is tiny. This proposal has impacted the IETF diffserv activity which is enabling QoS in ip networks by adding flags to transport headers, one of which is an active tag.

The second response came from the programmable network community who had for some time been looking to make networks more active with respect to operators [LAZ] and had progressed to a concept called “switchlets” [ROO] in order to avoid requiring operator intervention for all programmable changes. Switchlets enable clients to control their own VPNs by downloading their own control software onto a designated subset of the switch. This is only a partial solution as it only provides flexibility within a VPN for a single large customer. Programmable interfaces are being standardised in IEEE P1520 [BIS]

Smith and co-workers at University of Pennsylvania and Bellcore are working on a proposal (switchware [ALEX98]) which combines programmable packets, switchlets and a safe language (PLAN). This could be regarded as a realisable version of Tennenhouse, but only in the long term.

We have proposed a third alternative, known as application layer active networking [ALAN], which is perhaps the most immediately realisable. Similar proposals [AMIR,PARU] were described as active services. In these systems the network is populated with active nodes referred to as service nodes, or dynamic proxy servers. These can be thought of as equivalent to the http caches currently deployed around the internet, but with a hugely increased and more dynamic set of capabilities. They are logically end systems rather than network devices. This approach relies on redirecting selected packets into an application layer protocol handler, where user space programmes can be run to modify the content, or the communication mechanisms. Packets can be redirected using a single active packet tag in the transport layer header, or on the basis of the mime type in the application layer header. There is no need for additional flags or for any new standards (indeed many implementations use the ubiquitous http), and an arbitrarily large number of programmes, of arbitrary size can be used. The programmes can be selected from a trusted data source (which may itself be a cache) containing only well tested or specified programmes, and can be run without impacting router performance or requiring operator intervention since they do not impact the control kernel for normal packets. Programmes can be chosen to match the mime type of the content (in the

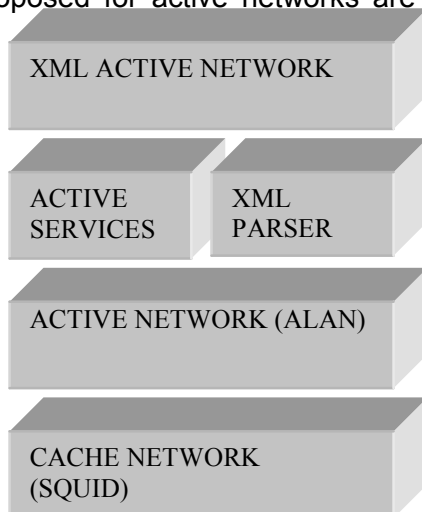
application layer header), so again no additional data or standards are required. Alternatively a more detailed specification can be supplied in xml metadata, if desired. There is a small performance penalty associated with the redirect operation, but this is acceptable for most applications. The major outstanding issue is the interactions between dynamically loaded programme, and this should be a priority for ongoing research.

There is a further proposal [CAO] that allows servers to supply cache applets attached to documents, and requires proxies to invoke the cache applets. Although this provides a great deal of flexibility, it lacks important features like a knowledge sharing system among the nodes of the network (It only allows interaction between the applets placed in the same page). The functionality is also severely restricted by the limited tags available in HTML (HyperText Markup Language). Most importantly the applets must be supplied by the content source and cannot necessarily be trusted. Clients do not have the option of invoking applets from trusted 3rd party servers.

Using mime-types (as in ALAN) provides more flexibility than HTML tags, but still restricts the range of applications that can be specified by content providers, as, different operations are often required for content with identical mime types. It is therefore necessary to find a better way to specify new services. XML provides a very promising solution, since the tags are extensible and authors can embed many different types of objects and entities inside a single XML object. For example policies describing resource and security requirements can be expressed and transferred with the object. In this paper we present a design for a modified ALAN based on XML, and describe how it could be used to provide a customer driven QoS routing capability . We also demonstrate feasibility of the use of XML by implementing and measuring a simple example service.

ALAN and XML

Our design is built in several layers and is based on existing technology. Figure 1 shows the architecture of the prototype. The first layer is a fully populated cache hierarchy, with caches placed at all domain boundaries and network bottlenecks. We envisage active nodes and caches being co-located since the optimal sites for most of the activities proposed for active networks are domain boundaries. In addition it is advantageous to



maintain a cache of programmes required by the active services at an active node and a web cache is a convenient implementation. For the prototype, we have used squid v1.19 [WESS] for the cache. The second layer and the upper layers constitute the core of our system and will be discussed thoroughly within this paper. An Application Layer Active Network Platform (ALAN) implements the active services. One of these services is an XML parser that provides the functionality to handle metadata associated with objects.

Figure 1 XML+ALAN Platform

The ALAN Platform is a Java RMI based system built by the co-authors from the University of Technology, Sydney in collaboration with BT-Labs to host active services. It provides a host program (Dynamic Proxy Server) that will dynamically load other classes (Proxylets) that are defined with the following interface: Load, Start, Modify, Stop [ALAN].

The platform provides a proxylet that analyses the HTTP headers and extracts the mime-types of the objects passing through the machine (HTTP Parser). After determining the mime-type of the object, the program chooses a content handler, downloads the appropriate proxylet from a trusted host to handle that mime-type, and starts the proxylet with several parameters extracted by the HTTP parser. Using this model, a wide range of interesting services can be provided. However, this original model cannot support the whole range of services we plan to implement. There is a need for additional data (not included in the HTTP headers) to manage interoperability among the services and to expand the flexibility and range of applications that can be developed. XML provides a mechanism to implement these improvements and appears a perfect complement to the architecture.

The BT co-authors built a simple XML Parser in Java that works in collaboration with a new HTTP parser designed to utilise all the Metadata needed in the active applications. The original HTTP Parser [ALAN] has been completely rewritten by BT in order to integrate the XML parser seamlessly into the processing.

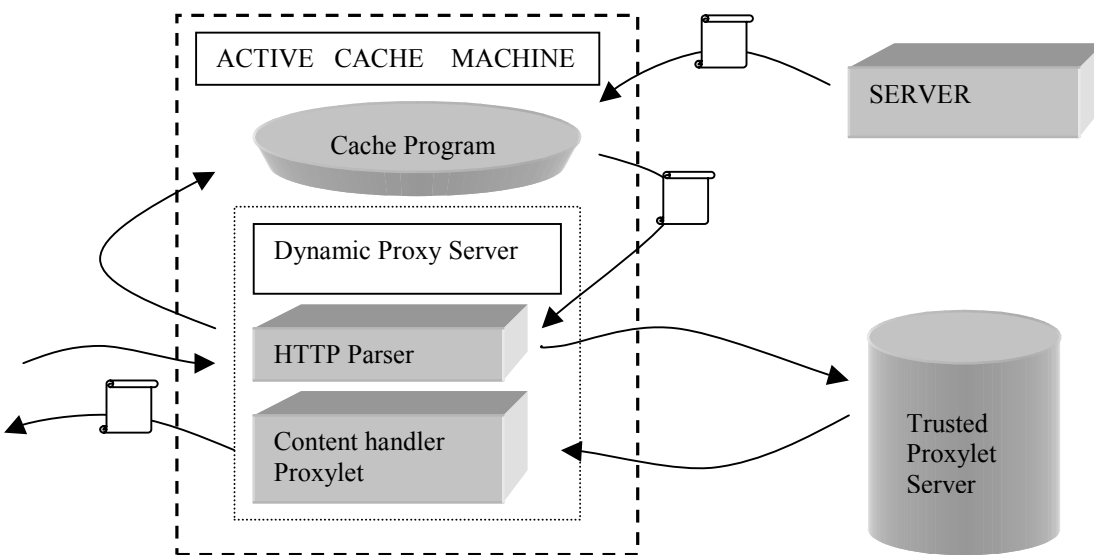


Figure 2 Functionality of an active Node

The functionality of an active node (figure 2) is described as follows. Upon the arrival of an object into the node, the HTTP parser examines the header and gets its corresponding Mime-Type. If the object is an XML object, then the XML Parser is called and it will extract the Meta-Data. The metadata specifies which proxylets should be invoked, in which order, with which parameters, and under what circumstances. The parser then makes the appropriate calls to the DPS, which loads the proxylets.

QoS ROUTING

Modern networks must optimise the management of communication among nodes that are interconnected by diverse and alternate paths. These paths may be based on heterogeneous, technologies with divergent properties. This makes smart path choice an essential feature in order to provide quality services (QoS). QoS management could be based on in-band datagrams (e.g. class of service based routing) or on out-of-band reservations (flow based routing). Both approaches have their adherents in the network research community. QoS-based routing has been recognised as a missing piece in the evolution of QoS-based service offerings in the Internet and is the subject of a range of standardisation efforts in the IETF including:

- Integrated Services - QoS - plus ISSLL
- Resource Reservation - RSVP
- Traffic Engineering
- Differentiated Service - Class of Service based

An interesting application that highlights many of the issues is the aircraft services application illustrated in figure 3. There are two basic services; communication with the

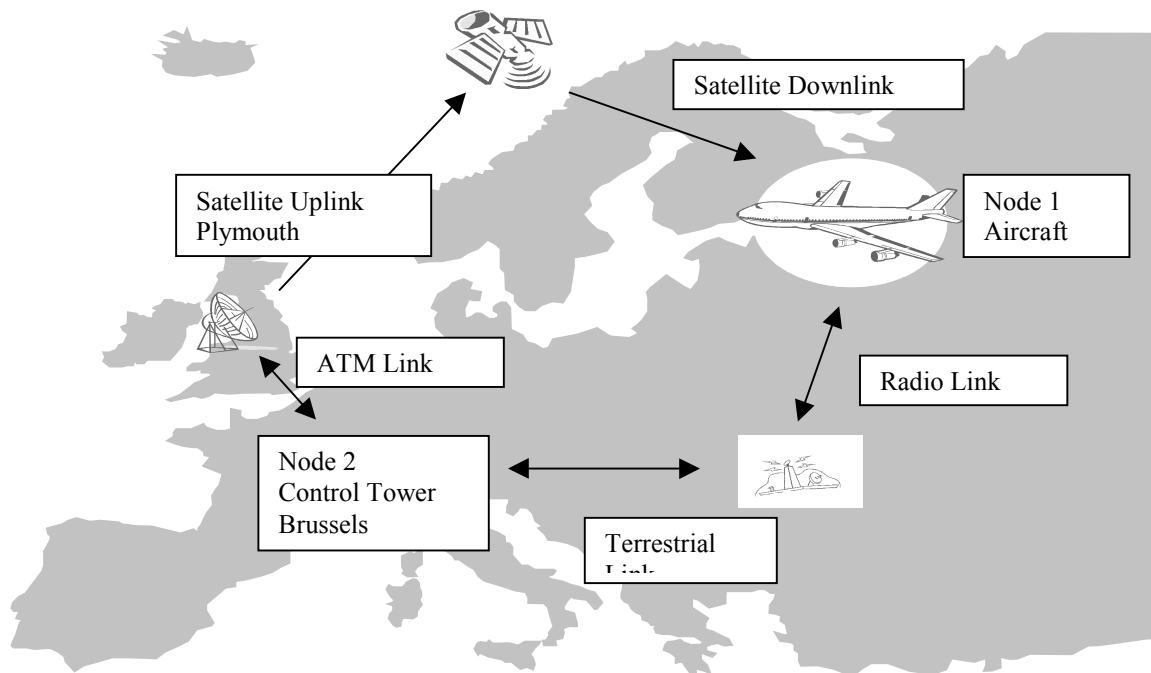


Figure 3. Aircraft application Alternate Path Routing Possibilities

flight deck and e-commerce/www access for the passengers. The aircraft (Node 1) has a 64kbit/s radio based bi-directional link to the control tower and a VSAT based downlink (2Mbit/s). The control tower can use wide-area connectivity to establish a high bandwidth link to the aircraft using ATM and Satellite combined. Path selection is performed at the Control Tower by examining the meta-information of the packets, deciding which is the most appropriate path and adding security if needed. There will also need to be QoS management in the plane to ensure life critical info from the flight deck is delivered into the bandwidth restricted downlink before any traffic originating from passengers.

To illustrate the power of an active information network of the kind we have described we have designed a QoS routing scheme which is suitable for the above application. The scheme requires no new standards and could be implemented entirely in the user space of network nodes. For this application QoS can be characterised in terms of bandwidth, latency, security, strength of guarantee and uni or bi-directionality. Current Internet routing protocols, e.g. OSPF, RIP, use "shortest path routing", i.e. routing that is optimised for a single arbitrary metric, administrative weight or hop count. These routing protocols are also "opportunistic," using the current shortest path or route to a destination [CRAW]. Our aim is to enable route decisions to be made on multiple metrics and fixed for the duration of the flow. For example a node could have a connection via landline with low latency, low bandwidth and high security, and a satellite connection with high bandwidth, high latency and low security. The choice of best route will be application specific. Given access to local route information obtained through link state adverts, nodes can make QoS decisions if the application requirements are also available. In our design the application requirements are expressed in XML metadata. The XML Metadata is rich enough to express the application layer requirements and force a correct choice. Using our scheme it also allows return traffic to be correctly routed using an appropriate proxylet at the head end of the satellite link

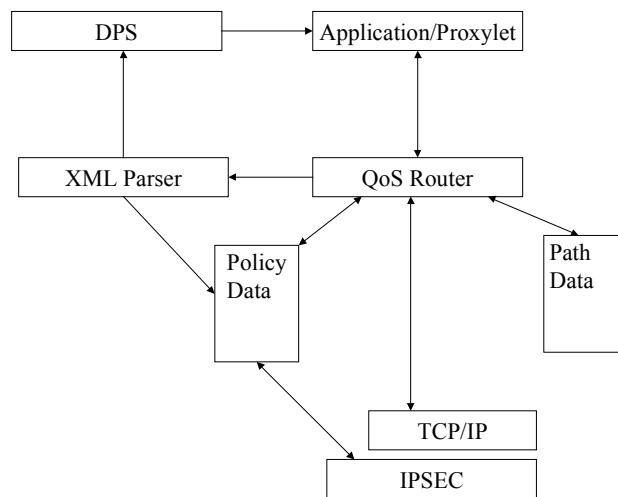


Figure 4 Active QoS routing node

The design of a QoS routing node is illustrated in figure 4. The path data is information about the QoS available on all local output addresses/ports. It is essentially a local routing table with added fields for measurement of delay, occupancy, loss rates etc. The measurements can be obtained by filtering link state adverts or using ping measurements. The policy data is a collection of policies regarding application requirements extracted by the XML parser from XML metadata. The policy syntax is based on the syntax for IPSEC security policies, where a set of fields are associated with a particular security association (or degree of security). This enables routing policies and security policies to be handled in the same way. For the routing policies the fields are; APP_TYPE, CONTENT_TYPE (MIME), BANDWIDTH, LATENCY, GUARANTEE, DUPLEX. The fields are associated with 6 tuples each containing the value of the field required by the associated content, and the priority (on a scale of 1 – 10) of obtaining that

value. The QoS router will intercept all socket requests from application layer processes, read all policies in the policy database relevant to that process, check the current path data and choose the output port which matches the most policy criteria. The criterion weighting is used to distinguish between alternates matching equal numbers of different criteria. Security criteria are regarded as mandatory – if no match is available the QoS router will either directly request user intervention (in an end system) or will deny the request (in an intermediate node). In the latter case the proxylet will request input from the session source or the preceding active node (which may just use an alternate route). In an initial implementation the QoS router would be a proxylet and would be invoked by direct calls from other proxylets requiring QoS routing services. We anticipate that, for performance reasons, if it proved popular it would be rapidly reengineered as a layered protocol intercepting all socket calls. In a retrieval session the content source would supply an XML object with its policies. The XML Object will be parsed, at any active nodes in the path, where the routing policies will be extracted and any other necessary proxylets will be started. The QoS router will then be able to choose the best available route for the associated flow. Any downstream active nodes can obviously perform further local route optimisations in a similar manner. For an interactive session the initiator would supply a session definition formatted as and XML object.

IMPLEMENTATION OF AN EXAMPLE & RESULTS

In order to get some preliminary performance measurements of the architecture we implemented advert rotation driven by XML metadata policies. The objectives were:

- Demonstrate the feasibility of XML policies.
- Show the weak parts of the implementation in order to improve releases in the future.

Studies show that advert banners that are dynamically rotated so the same HTML page can show different adverts for each request, are very popular. These dynamically created HTML pages are just slight modifications of an original template page. The changes usually consist of sets of graphics of the same size that will appear consecutively in the same position of the page. To achieve this, the server executes a cgi program that generates the HTML text dynamically. This dynamic behaviour tends to make this content un-cacheable. It is preferable to make simple dynamic pages containing rotating banners cacheable since this will;

- a) Allow a distributed service and eliminate the critical failure points.
- b) Improve the usage of the existing bandwidth.

This task requires Meta-Data that is not provided in the HTTP headers. XML will enable information like the rotation policy, the adverts that are going to be rotated, and perhaps a watermark to protect copyright, to be embedded.

As the object is requested and passes through the active node, it will be parsed in the http parser and then by the XML Parser. This analysis will extract the generic html

page that is going to serve as a static template for the rotated adverts, the list of images which should be rotated and the rotation policy. The information is used as a parameter list in the invocation of a rotator proxylet, which will download the objects as needed. Subsequent requests for the page will be passed to the proxylet by the cache at the active node, and the proxylet will execute the rotation policy. The complete XML specification is provided in Appendix 1.

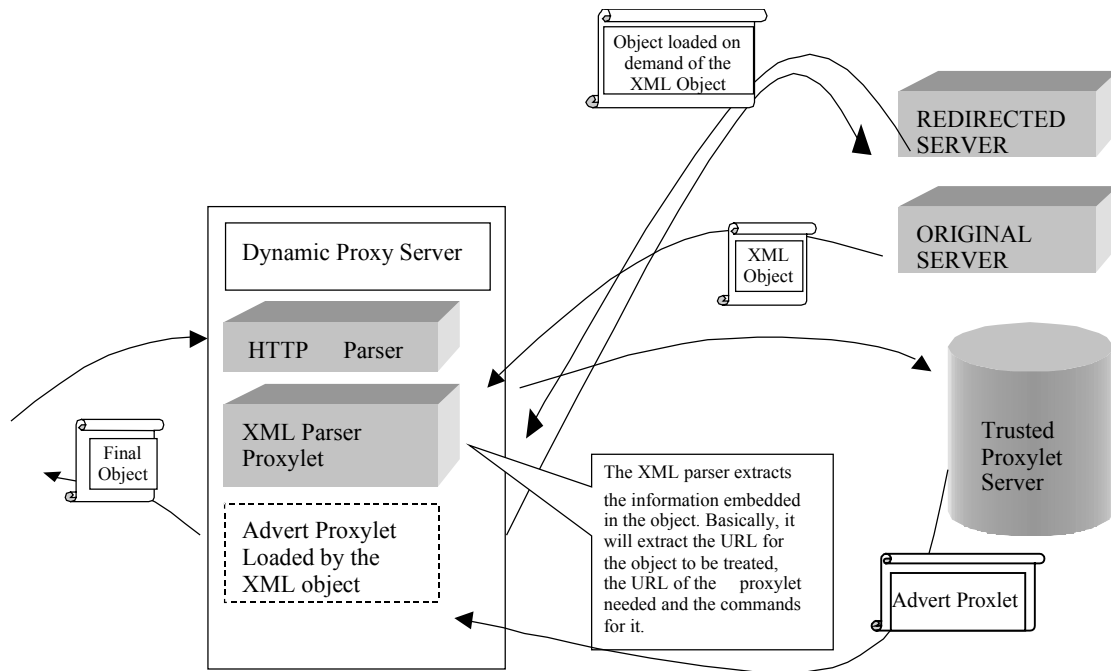


Figure 4 Advert Rotation Proxylet Functionality

Our experiment consisted of running an active node on a Sun Sparc Station 10 with 64 MB running SunOS Release 5.5.1. The Java version for both programming and testing was JDK 1.1.6. The active node program was started and was already running the Proxylets needed for HTTP and XML Parsing.

We conducted 20 experiments. For the first ten, the whole process ran each time a

First Request Time Distribution

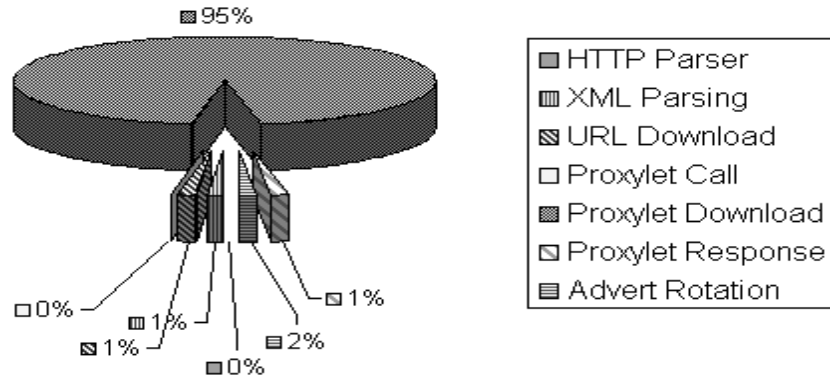


Figure 5 First Request Time Distribution

new advert rotation was requested, the advert proxylet was loaded. The subsequent ten utilised caching. When a new request arrived, a proxylet that was already running was used. We measured the times needed to accomplish the different tasks. The numerical results of these experiments are shown in the graph below. The proportional results are illustrated in the figures 5 and 6.

The analysis tries to show the times needed to perform the processes and tasks during the normal operation of the system. The functionality of these processes is described as follows:

- HTTP Parsing. Time needed to determine analyse the HTTP header and determine the Mime-Type.
- XML Parsing. Time needed to get the XML Object, parse it and extract all the Metadata Embedded.
- URL Download. Time needed to download the HTML.
- Proxylet Call. Time need to generate the query to load the Proxylet in our Active Node.
- Proxylet Download. Time needed to download and start the proxylet; it requires a lot of time because of the ALAN platform design.

- Advert Rotation. Time needed to perform the demanded task. In this case the advert rotation.

The graphic shows the average distribution of times during the first ten experiments. It appears that most of the time is spent downloading and starting a new service (setting up a new proxylet in the node). But once this service is set up, we can use it for further requests and avoid that undesirable delay.

Further Requests Time Distribution

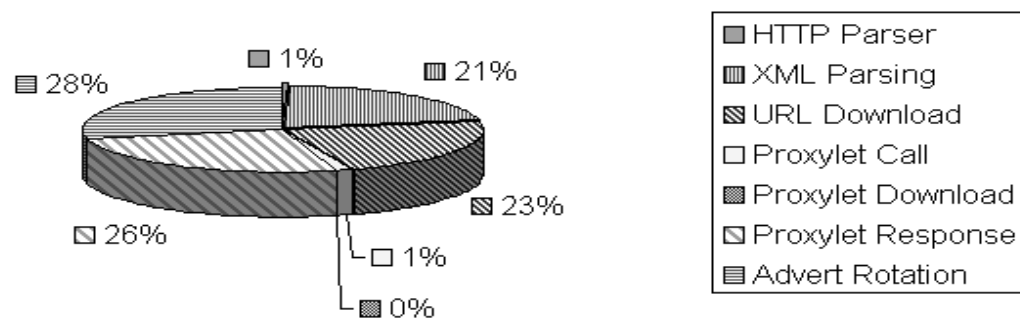


Figure 6 Further Requests Time Distribution

The graphic in figure 7 shows the results of the experiments when the service was cached. The proportion of time spent downloading the proxy has disappeared. The URL download time can vary depending the object to be downloaded and the bandwidth to the server. In our testing, all the objects are available in our LAN so we can expect greater values for this part of the process in wide area tests. However this increment will only be important for the first request, thereafter the URL object is cached and is made locally available.

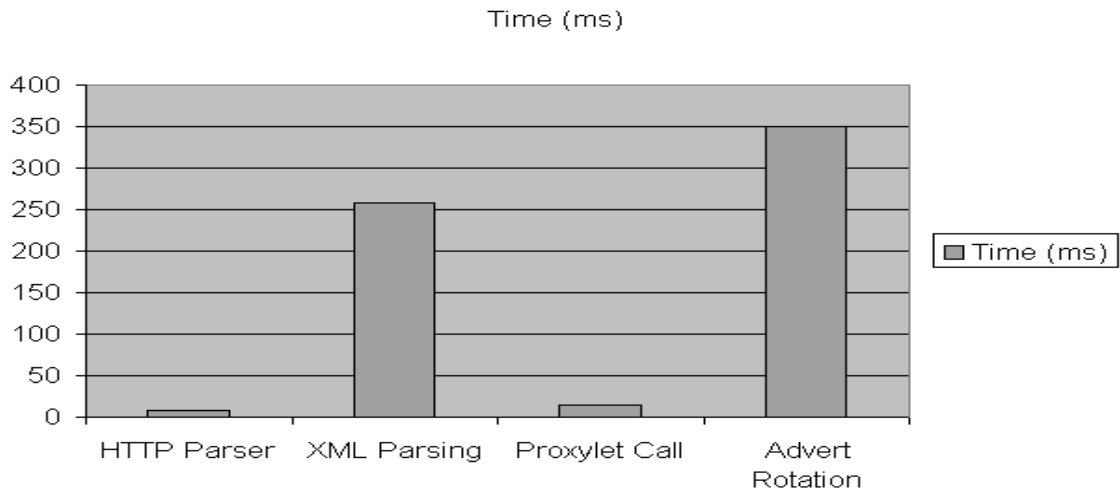


Figure 7. Time distribution in non-network processes.

The most important variable is the times due to the additional processing of the proxylets. It appears that the XML-Parser Proxylet and the Advert Rotator Proxyler are taking most of the time. Nevertheless the total delay is below one second. We can expect better results if a faster computer is used as a server with a non-interpreted language. However the purpose of this paper was to demonstrate the feasibility of active caching nodes based on XML and throughput was not a priority. This proto-type shows that it is possible to provide active services with delays of just several hundred milliseconds.

Finally, we present Figure 8 to illustrate the difference in the delay between the experiments that needed the proxylet to be downloaded and started and the experiments where the proxylet was already downloaded and running in the active node.

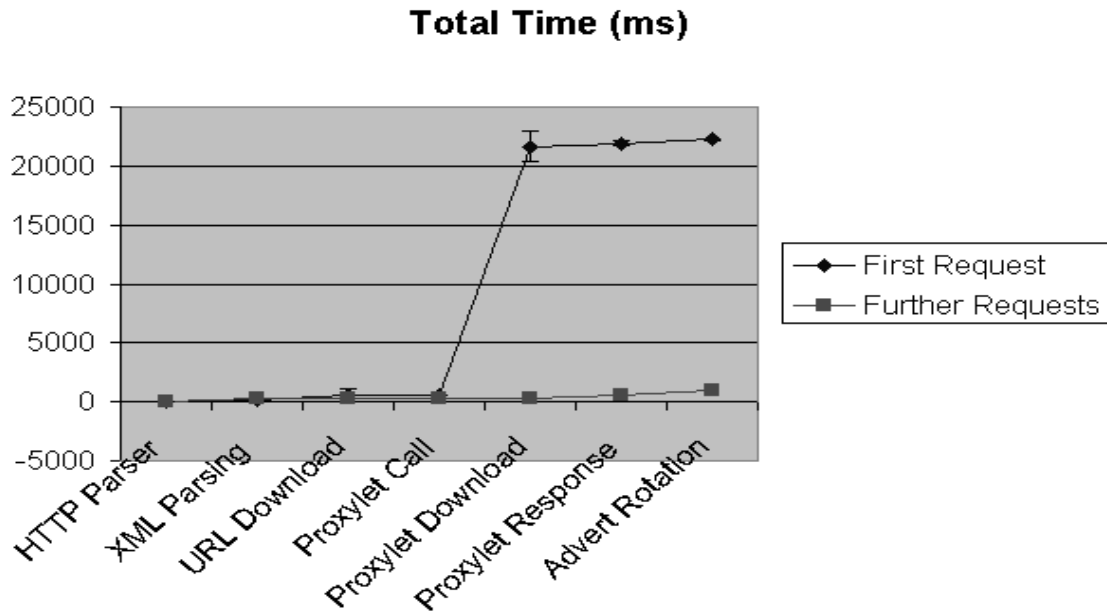


Figure 8 Comparison in global times of the first request with subsequent petitions

FUTURE WORK

In the immediate future we intend to build and test the QoS routing service outlined in this paper. In addition there are three major longer-term issues, which we are concentrating our efforts on solving;

- a) It would be beneficial to specify behaviour and other non-functional aspects of the programmes requested in the metadata, using a typesafe specification language. One possibility is to use SPIN, which is c-like and has some useful performance and time related primitives. The use of a language of this kind would provide greater flexibility and interoperability, and go some way towards solving our second issue.
- b) For complex services it will be necessary to invoke several proxylets. At present the question of how multiple proxylets interact, and how interactions such as order dependencies can be resolved, is open. We anticipate attempting to use metadata specifications (using the language from issue a)) to maximise the probability of avoiding problems.
- c) The performance and scalability of the DPS is currently far from ideal. The Sydney co-authors [ALAN] are addressing these issues with a new implementation, and we anticipate significant improvements will be available before our tests are complete.

CONCLUSIONS

Application layer active networks will play a crucial role in networked applications that can tolerate a delay of around a hundred milliseconds. They will extend the functionality and versatility of present networks to cover many future customer needs. HTTP caches help reduce the use of bandwidth in the Internet and improve responsiveness by migrating objects and services closer to the client. They are also ideally placed to evolve into the active nodes of the future. XML is a perfect complement to Application layer active networks based on http caches, since it will allow active nodes to be driven by enriched Metadata requests and at the same time will introduce the mechanisms for sharing knowledge between nodes. We have implemented a prototype, which demonstrates that XML offers greater flexibility and expressivity than HTTP tags or MIME types, without significant performance penalty. The performance of our system is not yet ideal, however, results can easily be improved by using a non-interpreted language and a more powerful server.

REFERENCES

- [ALAN] "Application Layer Active Networking" M. Fry and A. Ghosh,, Fourth International Workshop on High Performance Protocol Architectures (HIPPARCH '98), June 98.
<http://dmir.socs.uts.edu.au/projects/alan/prog.html>
- [ALEX97] "Active Bridging" Alexander, Shaw, Nettles and Smith, Computer Communication Review, 27, 4 (1997), pp101-111
- [ALEX98] "A secure active network environment architecture" D.S.Alexander et al, IEEE Network 1998
- [AMIR] "An active service framework and its application to real time multimedia transcoding" E.Amir, S.McCanne, R.Katz, Proc SIGCOMM '98 pp178-189
- [BIS] "The IEEE P1520 standards initiative for programmable interfaces" J.Biswas et.al., IEEE Comms. Oct 1998 pp64-72
- [CAO] "Active Cache: Caching Dynamic Contents (Objects) on the Web" P. Cao, J. Zhang and K. Beach Proc middleware '98 (Ambleside).
- [CRAW] "A Framework for QoS-based Routing in the Internet". E. Crawley. R. Nair. B. Rajagopalan. H. Sandick. Copyright (C) The Internet Society (1998). <ftp://ftp.isi.edu/in-notes/rfc2386.txt>.
- [ERIK] "MBone - The Multicast Backbone", Eriksson, Hans, INET 1993
- [LAZ] "Programming Telecommunication Networks" A.Lazar, IEEE Network Oct 1997 pp 2-12
- [PARU] "Active Network Node Project" G.Parulkar et.al Washington University St Louis
- [ROO] "Tempest: A framework for safe programmable networks" S.Rooney et.al., IEEE Comms Oct 1998 pp42-53
- [TENN] "Towards an active network architecture". Computer Communication Review, 26,2 (1996) D. Tennenhouse, D.Wetherall.
- [WAK] "Designing a Programming Language for Active Networks" I.Wakeman et.al. HIPPARCH '98
- [WESS] "Configuring Hierarchical Squid Caches", Duane Wessels AUUG'97, Brisbane, Australia.
- [XML] Extensible Markup Language (XML) W3C Recommendation 10-February-1998
<http://www.XML.com/aXML/testaXML.htm>.

APPENDIX 1

```
<?XML version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE novel PUBLIC "-//ACTIVE-XML-VEL98//DTD ACTIVE-XML//ES" "activeXML.dtd"
[]>
<dynamic initiator>
<front>
<title>active-XML-example</title>
<author>Luis Velasco</author>
<revision-list>
<item>XML version 1998</item>
</revision-list>
</front>
<body>
<programms to load>
  <program>
    <name> activebann.Activeban </name>
    <orders>
      <load>
        <trusted program server>
          http://midway/proxylet/activebann.jar
        </trusted program server>
        <host machine>
          midway.drake.bt.co.uk:1998
        </host machine>
      </load>
      <start>
      <arguments>
        banner1 banner2 banner3 banner4 RANDOM
      </arguments>
      </start>
    </orders>
    <orders_other_caches>
    </orders_other_caches>
  </program>
</programms to load>
</redirect object>
</redirect object>
</body>
</dynamic initiator>
```

APPENDIX 2

This sheet shows the results in milisecond for the different experiments deployed in the last part of the paper. It is important to note that the figures are in milliseconds and that the Proxylet download time only includes the values for the first ten experiments, then as this step is skipped, this time is reduced to zero.

	HTTP Parser	XML Parsing	URL Download	Proxylet Call	Proxylet Download	Proxylet Response	Advert Rotation
Average	7.6	257.4	278.0	14.9	21735.4	317.2	350.8
Standard Error	18.7	180.7	563.6	1.8	1307.6	198.5	63.1