

ALPINE - Application Level Programmable Inter-Network Environment

Ian W Marshall, James Cowan, Jon Crowcroft, Mike Fry, Atanu Ghosh, David Hutchison, David Parish, Iain Phillips, Nat Pryce, Morris Sloman, Dan Waddington

ABSTRACT

Future requirements for a broadband multimedia network are likely to be very different from those of today. Three key changes 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. In the management of multiservice networks project, BT and its academic partners are developing key middleware components required to support application layer active networking.

1. INTRODUCTION

The characteristics and behaviour of future network traffic will be different from the patterns 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 asynchronous transfer mode (ATM) or internetwork protocol (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 quality of service (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.

One of the great strengths of the Internet is that a number of its functions are loosely coupled (naming, addressing, routing, congestion and flow control, to name but a few). This means that partial failures of the network do not bring the whole system to a grinding halt. It has its downsides, but appears to be a good basis for doing things. BT has funded a programme called the 'Management of Multiservice Networks' (mmn) University Research Initiative since 1994 which is exploring how we reflect this design philosophy in a *programmed environment*. The programme has developed a number of key active network middleware components, including an active server, resource controllers and monitors, and management policies. This paper introduces active networks and justifies the choice of application layer active networking. It then describes the components developed to date by the academic partners, and discusses the ongoing research that is required.

2. Active networks (Tennenhouse)

Active networking was originally [Tenn] a proposal, by Tennenhouse at MIT, to increase network flexibility by adding programmes to the packet header, that are intended to run on network devices that the packet encounters. 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 576 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 and designing programmes which safely invoke other programmes is not currently possible. Since it will be extremely hard to create interesting programmes in a language which is simple enough to enable complete 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 run. However, the routers do not receive the acknowledgement packets that would terminate the programmes as the acks are routed elsewhere after the clients location has changed.
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.

3. 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 Internet engineering technical forum's (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. Switchlet development at Cambridge has been partly funded by the mmn project, but is not covered in this paper since it only provides flexibility within a VPN for a single large customer and is not currently a significant part of our design for an active services infrastructure. 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.

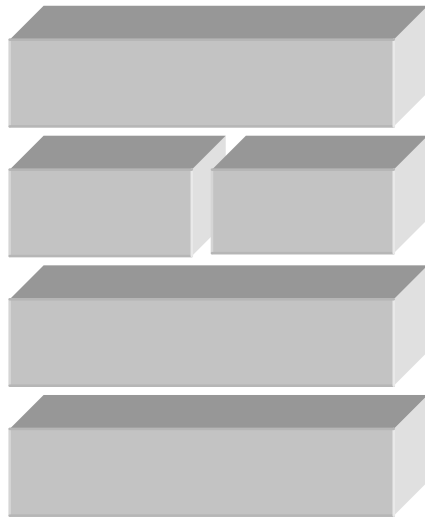
We have proposed a third alternative, known as application layer active networking [Alan], which is perhaps the most immediately realisable. Similar proposals [Amir,Paru] were described as active services. In these systems the network is populated with active nodes referred to as service nodes, or dynamic 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 hypertext transfer protocol (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 programmes, 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.

4. ALAN (UTS & BT)



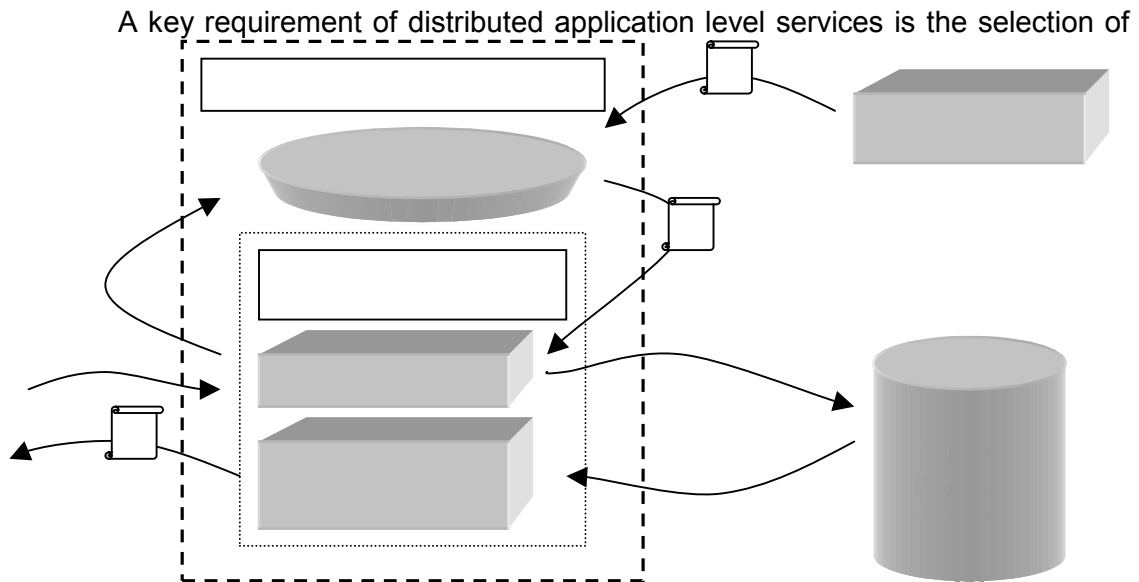
Our design is built in several layers and is based on existing technology. Figure 1 shows the architecture of the prototype. The first layer is a fully populated cache hierarchy, with caches placed at all domain boundaries and network bottlenecks. We envisage active nodes and caches being co-located 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.

The ALAN Platform is a Java RMI based system designed to host active services. It provides a host program known as the DPS (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. However, 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. We have built a simple XML Parser in Java that works in collaboration with the HTTP parser, and is designed to utilise all the Metadata needed in the active applications.

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.



the appropriate server from a pool in a scalable way. A Proximity metric can be delay, or throughput, and may reflect both network and application level performance. In a world where there are many DPS's distributed around the world a mechanism is required to discover the optimally located DPS's for a particular task. We believe that a mechanism analogous to a routing protocol will need to be adopted to allow the discovery of a DPS. In this work, we would build the proximity service as a unification of two separate services, one host based and the other router based. The router based service would be based on ideas from the internet routing protocols OSPF and BGP (e.g. Link State adverts from OSPF). We can also relate DPS location to the caching hierarchy, and use application layer host based routing techniques developed by the caching community to solve this problem. We are thus still operating at the application level, not touching the underlying transmission system, which may be IP, ATM or whatever is available. The aim is real time construction of overlays that are self-organising to meet the needs of the application and its instantaneous environment.

DPS discovery will need to be performed based on a number of metrics including:

- Allowed domains
- Closest host (in terms of bandwidth or latency)
- Some fault tolerance metrics
- Processor availability (to support heavily processor intensive transcodings)

One of the perceived problems with allowing arbitrary proxylets to run on DPSs is the possibility of undesirable interactions. Rather than attempting to exhaustively test a proxylet, it may be sufficient to force it to live within tightly managed resource constraints so that it cannot affect other proxylets run by other customers. It is then the responsibility of the user or author of a proxylet to verify that a particular proxylet behaves correctly. Of course it should still be perfectly reasonable for a proxylet to run for extended periods of time in some contexts. A mechanism could be developed to accept some form of electronic cash in order to continue to run a proxylet. The owner of a DPS will not mind that a local resource is being consumed if s/he is being suitably recompensed. In order to address the management problems the project is now attempting to build mechanisms for proxylet admission control and policing, based on the work of all the partners. This involves the definition of proxylet metadata, and permits DPS load balancing. We are also developing policy based QoS management via feedback monitoring to an adaptive streaming proxylet.

5. Dynamic stacks and policies (Imperial College)

5.1 Dynamic stacks

Imperial College have built a flexible distributed programming environment (DPE), *Regent*, that complements the work on Application-Level Active Networks (ALANs) performed at UTS [Alan], by enabling the protocol stack to be configured to match the QoS requirements of each proxylet using it. *Regent* differs from existing DPEs, such as CORBA ORBs [OMG95] or Microsoft's DCOM [Rog97], in two respects: multiple interaction styles between application components and dynamic protocol stacks.

Application components are not limited to object-based request/reply interactions. A language has been developed (Midas [PC97]), which permits the programmer to write an interaction specification in terms of asynchronous messages and state machines. This allows the succinct specification of the various interaction styles required by a large distributed system, such as request/reply, event dissemination, bulk data transfer or media streams. Midas syntax is similar to the CORBA Interface Definition Language (IDL). It uses IDL syntax for constant, type and module definitions and replaces object interface definitions with definitions of interactions following the model described above. Midas also allows the definition of generic types and interaction styles that are themselves parameterised by type, and so does not need an *Any* type to represent untyped data.

Midas specifications are compiled into implementation language constructs, such as Java classes, that define the message interfaces and provide distribution transparency via proxy objects. Midas allows interactions to be defined independently of the implementation language and simplifies the task of the programmer by generating support for distribution transparency.

The runtime support generated from Midas specifications interfaces with the *Regent* transport subsystem. This subsystem provides an object oriented framework for implementing transport protocols. The framework takes a component-based approach: lightweight protocol layers encapsulate small aspects of protocol functionality and are composed into protocol "stacks" (actually directed graphs) in order to implement the required transport protocol.

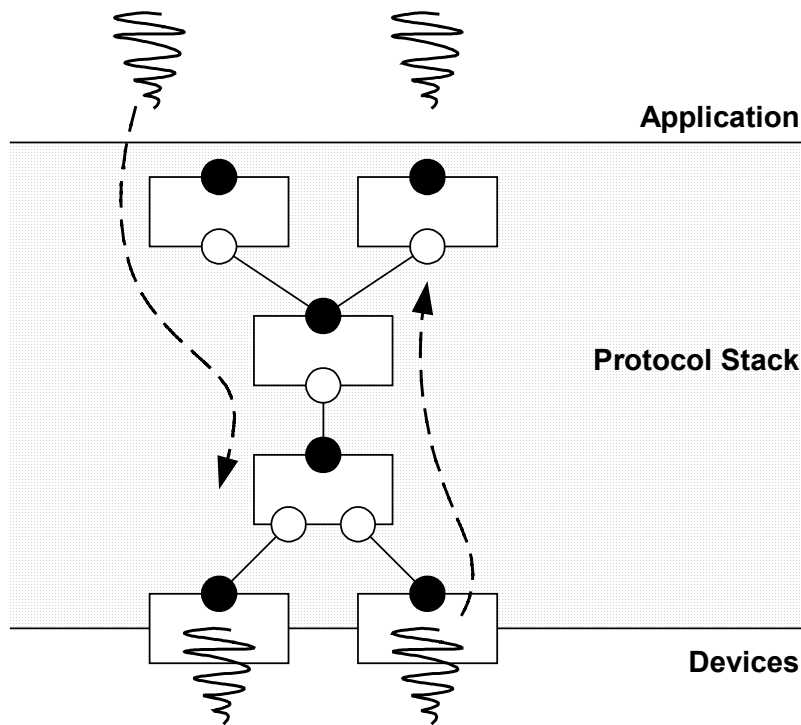


Figure 3 Protocol Stacks

Protocol stacks (see Figure 3) can be identified by composite names that specify the layers within the stack. These names can be "elaborated" by the Regent runtime to dynamically load implementations of the named layers into the node, instantiate layers and compose them into the stack described. References to distributed services contain these stack names, allowing servers and clients to evolve independently, making use of new transport protocols as they become available.

A protocol layer can also provide control interfaces through which higher layers can control the operation of the protocol. Control interfaces can also expose events to higher layers. To be notified of events occurring beneath it in the stack, a protocol layer must implement a compatible "event listener" interface and register the interface to be called back when events occur. The model of attributes and events used by control interfaces is similar to that used by the Java Beans component framework.

A useful type of protocol layer is the *virtual protocol*, that does not modify the packets passing through it. Virtual protocols can be added to a stack without affecting its compatibility with any remote system communications. Virtual protocols that expose control interfaces and events are the basis for configuring monitoring and management support into a node's communication subsystem.

5.2 Policies

Imperial college have also developed a policy notation and support tools which permit the specification of *authorisation* and *obligation* policies [Slo94, Marr96, 97]. Authorisation policies specify what activities a manager is permitted or forbidden to do to a set of target objects and are similar to security access-control policies.

Obligation policies specify what activities a manager must or must not do to a set of target objects and define the duties of a manager. Obligation policies define actions that are triggered by events. Policies are interpreted by automated manager agents, and so the behaviour of the agents can be modified *dynamically* by changing policy rather than re-coding. The policies thus provide a constrained form of 'programming' of automated agents to change management strategy without shutting down the management system.

The Darwin Architecture Description Language (ADL) is being used to specify the ALPINE components required for a particular application. This will give a declarative specification of what proxylets are required, what dynamic proxy servers they should be loaded onto, what protocol stacks are potentially required and how they should be configured. It is envisaged that Darwin statements will be embedded in the XML metadata. The Darwin language allows a component's implementation itself to be distributed. This would allow the specification of proxylets that span multiple DPS servers, making best use of available communication, computation and storage resources.

The advantage of an approach based on the use of Darwin is that more checking can be performed at design time to detect potential incompatibilities and problems. Of particular benefit is the fact that architectural descriptions are an ideal skeletal framework for other support tools e.g. graphical design tools, compilers, behavioural and performance modellers, debuggers and application management software. The Midas language integrates with Darwin to allow the modelling and analysis of designs that take the component behaviours, interaction protocols, and transport layers into account.

The proxylets required to perform a particular application and the protocols they use may need to adapt to events such as component failures or changes in the Quality of Service during a long-running interaction. The obligation policies provide a flexible means of specifying how the systems should adapt to these changes. The advantage of the obligation policies is that they are interpreted and so can be dynamically disabled/enabled or replaced without stopping the manager agents i.e. the management policy of the active networking can be dynamically changed as well as the proxylets themselves. The research objective will be to see whether the event based obligation policy rules can be used to control proxylets and protocol stacks.

As part of the investigation of policies to manage transport protocol stacks, we intend to insert virtual protocols into a binding's stack to monitor the actual quality of service available to the binding. If the QoS varies beyond some predetermined threshold, control events can be generated. Management agents within the node can use these events to trigger policies as to how to react to changing QoS levels. Potential actions could include fine-tuning the operation of the protocol stack via the control interfaces exported by layers in the stack, replacement of the stack with another which makes better use of network resources, or the triggering of application-level changes, such as changing the media encoding used or selection of a different Dynamic Proxy Server from those available in the ALPINE framework.

Another issue addressed by the policy framework is security. Authorisation policies can be used to define what proxylets or stacks can be loaded by a server, who can install proxylets on a proxy server etc. Obligation policies can be used to specify actions to be taken in the event of security violations.

6. Server resource controls (Lancaster)

Lancaster University's work falls under the broad title of "End-system QoS Management". Management of distributed multimedia application sessions involves three distinct processes; *QoS specification*, which captures application level QoS requirements; *QoS mapping*, which translates user-level requirements to finer grained network and operating system level requirements; and *QoS assurance*, which is the maintenance of the end-to-end service through resource monitoring, policing and adaptation. The Lancaster contribution consists of the development of a distributed component infrastructure, incorporating a multitude of mechanisms to facilitate end-to-end QoS management. With regards to ALPINE, the support is focused upon the management, allocation and protection of end-system resources according to the individual application requirements (including those of proxylets) maintained within the end-system. We have principally concentrated on providing suitable tools, abstractions and middleware mechanisms for the support of these processes in the end-system. Figure 4 illustrates aspects of the end-system that are directly addressed by our work (illustrated as shaded areas).

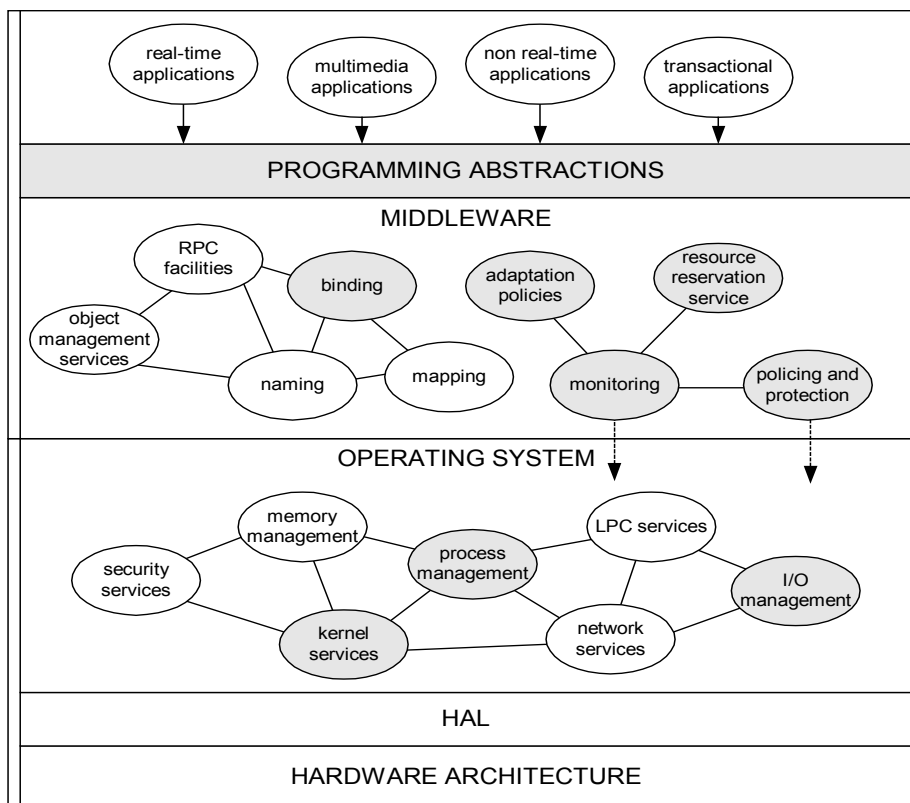


Figure 4. QoS Management architecture

Many current distributed object environments, including CORBA [OMG91] and DCOM [Mic95], do not support abstractions for continuous media interactions which enable the QoS requirements to be mapped to object implementations. To aid the task of mapping we introduce the concept of hierarchical binding. Hierarchical binding means that a binding object 'delegates' the task of mapping from point-to-point requirements to further sub-bindings. This approach to binding means that each binding object need only retain enough resource mapping information to break down a point-to-point requirement to two or more finer grained requirements. For instance, an end to end service binding can be reduced to two bindings in the end-

systems and one binding in the network. All services in our architecture are essentially provisioned through one or more hierarchical, distributed bindings between components (such as proxylets or user applications). Binding objects automate the selection, configuration and control of resources between two or more interfaces.

Once a service has been created, we have the problem of how we control, manage and dynamically adapt the service during its lifetime. The hierarchical binding tree is an ideal mechanism for the support of QoS monitoring and QoS management. Other researchers [Nah95] have proposed the introduction of a single QoS manager component, where the QoS manager is responsible for QoS assurance through resource allocation, monitoring and maintenance in both the end-system and the network. However, we believe that such a component would require a huge amount of mapping and management knowledge to support large scale distributed applications, and that service management through a single entity is too centralised and severely inflexible. Our solution is to build the QoS management into the bindings themselves, i.e. to incorporate the management functionality into the communication mechanisms between one or more interfaces. Expressing the requirements as policies in proxylet metadata, and using a dynamic protocol stack enables the management to be associated with the binding in precisely this way. Each binding is responsible for a point-to-point contract and must make every effort to maintain this agreement. The binding may employ control invocations on lower level entities or even carry out component re-configuration. If the binding is unable to sustain an agreed contract, then it should signal QoS failure or degradation to its parent binding. If the binding object is the service binding object, i.e. the root of the binding tree, then feedback is given directly to the client application. The client application may then decide on an alteration of service requirements. One of the issues of such an approach to the management of a service is that the component, which is responsible for the degradation of service, carries out the necessary action to resolve the problem instead of relying on other components within the session to take action.

In order to suitably control and co-ordinate the utilisation of end-system resources between multiple applications, we need to introduce services for resource reservation, admission, monitoring, policing and adaptation. Mechanisms for these processes are provided by the End-System Resource Manager (ERM) which has been developed as a multimedia, middleware system service (Figure 5). We have employed a form of 'loose' resource reservation. This means that resources are reserved, admitted and released, as with standard approaches to resource reservation, except the system cannot offer firm guarantees on the availability of resources even if reservations have been made a priori. The system therefore also uses dynamic QoS management to respond to unforeseen variations in resource availability. The infrastructure offers a best effort management service, and will take all possible measures, including the use of policing and adaptation, to ensure that reservations which have been admitted by the system are suitably respected. The main reason for the use of this approach is because of the relatively unpredictable availability of resources within a non real-time priority based operating system such as Windows NT. Furthermore, even if we can introduce sufficient policing and guaranteed resource scheduling, we cannot expect legacy applications or the operating system itself, to follow a mandatory reservation process.

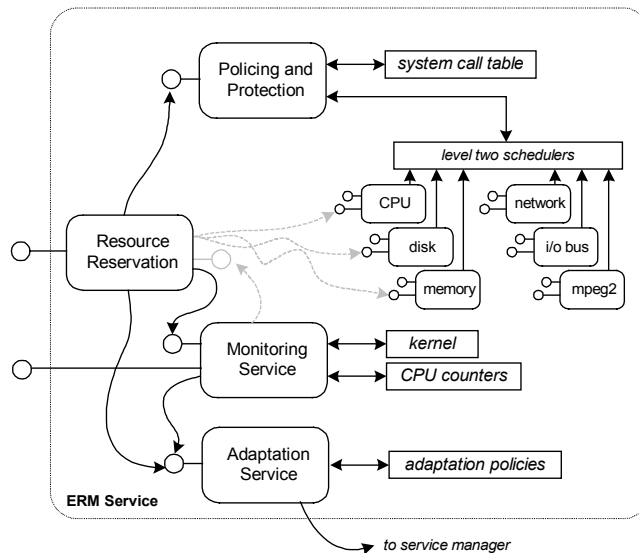


Figure 5. End-system Resource Manager

Each resource managed by the ERM is encapsulated as an extensible 'resource module'. These modules are responsible for the fine grained allocation and scheduling of resources according to earlier reservations the client made via the ERM. Generally, each of the resource modules requires access to scheduling and control mechanisms at the lowest level, i.e. within the operating system executive. Mechanisms we have prototyped include kernel modules for resource policing (interceptors for disk i/o and memory allocation, and specialised thread scheduling for processor management). We have also arranged the scheduling in a split level scheme. The first level of scheduling fulfils the role of translating relatively coarse grained requirements into requirements which can be released by the respective low level, or level two, schedulers. This approach means that the complex task of translating QoS/resource abstractions into meaningful scheduling requirements is only carried out at the point of admission thus avoiding any degradation in general system performance. Furthermore, split level scheduling means that we can introduce application specific reservation abstractions without the need to adjust the underlying level two schedulers.

In conjunction with the resource reservation services provide by the ERM, the service also incorporates services for QoS monitoring. To do this the ERM acts as a retrieval agent for extensible performance counters in the operating system. These counters are generally associated with the resources in the end-system and the actions they perform. Furthermore, application specific counters may also be introduced. This is particularly useful for monitoring activities in continuous media processing, such as missed frames or frozen buffers. The monitoring information can be retrieved by a remote client, via the ERM. This feedback can also be used by the run-time management mechanisms to automate responses to fluctuations in available resources or alterations in application level requirements. Within the work we have developed a general model of QoS adaptation [Wadd98] which is used to co-ordinate the balance of resource utilisation across the end-systems and the network. Adaptation decisions are made in response to negative feedback from resource monitoring in the end-system, and may take the form of resource control or end-to-end reconfiguration. Currently adaptation policies are statically defined by the system administrator and cannot be adjusted dynamically, aspects of dynamic policy specification will be addressed in future work.

7. Network resource controls (UCL)

UCL is engaged on work on the management of protocol stacks within end systems. Resource reservation protocol (RSVP) can ensure that time critical applications receive the resources they need, but only at the cost of using a non-scalable amount of router cpu resource. Instead of controlling traffic within the network where CPU resources are scarce, we aimed to control how much traffic enters the network at its edges. The work is focussed around transport protocol performance control through remote access from a management centre to the key performance parameters in a protocol. The system can operate in a client and in principle, in a server, and can be added without re-building (or re-compiling) the application or operating system protocol stack. Remote access from Java RMI, Corba RPC and SNMP/CMIP are all possible. The system is intended to use feedback from the Loughborough work which makes available detailed network performance statistics (see below). The basic idea is to instrument the TCP-IP stack so that any application can be rate controlled dynamically via network management. Instead of embedding congestion control algorithms in the TCP-IP stack itself, we can apply different rates to different applications depending on their Quality Of Service requirements.

We have developed a prototype of the protocol stack management system. The purpose of the system is to provide a standard network management interface to the protocol stacks running in Windows NT computers; a network management agent running on each computer monitors and controls the traffic running through an instrumented Winsock2 TCP-IP stack. Network managers can use the system to control transparently how much network bandwidth an Internet application can use – in this context, transparently means that the controls can be applied without any changes to the Internet application. Bandwidth controls can be applied to a group of hosts or a single host, to a group of processes within a host or a single process and to a group of sockets within a process or a single socket.

The protocol stack management system consists of a number of components, as depicted in Figure 6:

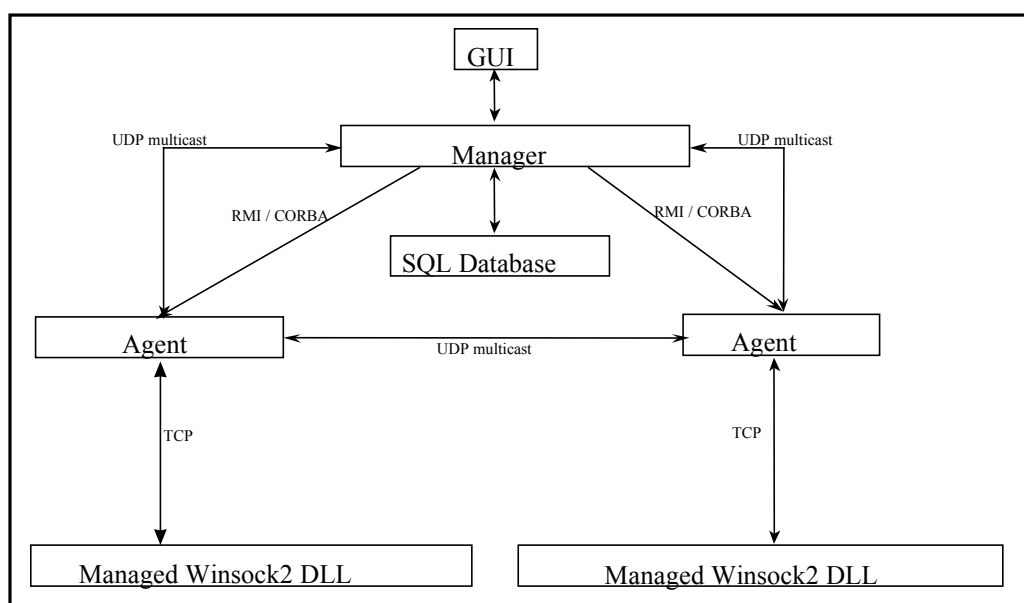


Figure 6. Stack management system

An instrumented Winsock2 TCP-IP Dynamic Link Library (DLL) monitors and controls traffic at the lowest level of the system. The DLL reports how much traffic is being sent on each socket and allows a manager to block and rate control sockets. A Java agent running on each machine controls the Winsock2 TCP-IP stack. It presents a CORBA interface that allows network managers and applications to decrease and increase throughput rates for any socket. It uses a purpose built reliable multicast protocol to send monitoring information; multiple managers and applications can retrieve monitoring information without swamping the network with management traffic. A central manager with a GUI allows network managers to communicate with the agents on each computer and control how much traffic each host and process can inject into the network. A relational database system stores monitoring information gathered from each agent. It logs how much traffic each host, process and socket has generated.

Recently the focus of our research has been to deploy the protocol stack management and enhance it so that it can be used to do Internet server analysis and control. We are deploying the system on an array of servers at BT Labs. We have extended the monitoring capabilities of the system so that it can do traffic analysis on a per minute, per hour, per day or per month basis. We can also present traffic statistics on a user basis rather than just a host basis. The system could be extended to do Internet pricing which is likely to be an effective traffic control measure.

In the first instance the stack manager serves the purpose of limiting the amount of network resource available to proxylets, and preventing interference between different flows originating from proxylets. In the future we would also like to use it as the basis of a low level implementation of the DPS that enables the addition of transport layer services as flexibly as application layer services. Current IP routers provide a very simple forwarding function, together with a complex route exchange and computation function. As we add further services such as integrated and differentiated services, mobility support, security and so forth, the control functionality concerned with both the routing **and** the forwarding function become more complex. Proponents of "strong Active Networks" claim that, provided there are sufficient safeguards, distributed programs may be dynamically updated to create new services "on the fly". In this area of work, we require a parsimonious approach to adding programmability to routers. The high-level virtual machine (VM) used in ALAN will need to be reengineered to operate at a lower level in the operating system, whilst still supporting the high level proxylet interface. Based on the work at Sussex University [Wak], we will design a low level VM that would interact with the baseline (unicast, multicast, mobile, QoS) routing and forwarding functions, and allow combinations of low level operations to make up higher level programs.

8. Internet monitoring (Loughborough)

In order for the management and control mechanisms outlined above to operate correctly and fully support the DPS, it is necessary to have access to dynamic information about the state of the network at remote sites. The DPS is intended to establish proximity of nearby active nodes through delay based measurements dominated by the response time of the remote server. However, to maintain QoS we believe that the DPS will need access to path level metrics such as

bandwidth and service classes (QoS) available on each path to nearby active nodes. It will also need information on current load (per QoS class), loss rate and burstiness for each path. Such information is hard to obtain in the current internet. Loughborough University have been funded by the mmn URI to develop network performance measurement tools which could evolve towards satisfying our requirements. The work commenced with the development of a distributed architecture for network performance measurement which is generally applicable to any packet or cell switched network. This architecture consists of a number of (relatively simple) remote stations whose task is to generate and receive test messages via the network. These messages are generated at the ultimate command of a control station whose complexity is greater than that of the remote stations and includes the ability to retrieve, store and process the measurement data. A suite of software tools associated with this station can produce various summaries of network performance for research or operations based users.

An instance of this architecture [Phil] has been deployed on BT's SMDS network where IP packets are generated, received and analysed in order to help understand the performance of the network at higher levels of abstraction than merely that of basic connectivity. This system has now been running operationally for in excess of 2 years at the time of writing.

The system architecture has also been enhanced using the object orientated paradigm. The components (e.g. remote stations, data bases) etc are now viewed as objects allowing greater flexibility in future deployment and development. In particular it allows a less centralised model in which all active nodes can be control stations.

Considerable work has been conducted on enhancing this object-orientated architecture [Bash]. It is a fact of network performance analysis that the basic monitored data are often repeatedly processed in different ways as the analysis task develops. This becomes a considerable processing challenge to the system and its database, and is also inefficient as basic processing operations are often repeated. Work has led to the development of a database using intermediate information structures to allow subsequent reuse of the data to be efficiently and safely achieved when multiple user information requests are processed.

The architecture has allowed more advanced performance monitoring activities to be considered. Two of these developments are referred to as Adaptive Monitoring and Automatic Incident Reporting (AIR).

Adaptive Monitoring allows a monitoring schedule (a test) or a routine processing operation to be modified automatically in the light of new information. An example of this would be the automatic detection of an increase in average network delay at a certain time of day. This indicates a need for more detailed measurement of other metrics of interest. An additional test on the suspect route would then be automatically initiated. This avoids a need to continually test on all the metrics of interest and minimises the test traffic in the network.

AIR is a mechanism to allow low level test data to be automatically processed into a higher level of useful information for network operators or application layer processes such as the DPS. In many cases, the users of a network are mainly concerned with the current and future well being of the network. This can often be

deduced by an intelligent interpretation of packet or cell delay and loss characteristics. The AIR system uses a combination of techniques, including statistical matching to identify network routes and times when unusual characteristics of the network exist. The power of AIR is that it highlights these effects automatically, saving the human user the time consuming task of manually inspecting the performance data generated by the monitoring system, and saving programmers the task of building expensive data filters into application layer processes using the data.

9. Discussion

The work outlined here has all been demonstrated on hosts within BT labs, much of it concurrently. The final deliverable for the current URI contract is an integrated deployment on LEANET [ref] during the summer of 1999. This will be followed by further collaborative work on the outstanding research issues. Initial deployment of active service functionality in public networks is expected before 2002 with commercial services following soon after. The first deployment is expected to be of simple proxylets that cannot invoke further proxylets (i.e. only one per service) and operate exclusively in the application layer. The system will be a fully developed, high performance version of the current ALAN based around http. We anticipate that by around 2006 there should be complex interacting proxylets and a low level VM for transport layer services. Information interoperability will be based on http-ng, and the URN scheme. There should also be a complete portfolio of information layer services such as proximity detection, name resolution and directories. By 2010 it is possible that active node hardware will be flexible too, and certain that it will be a tighter integration of routing, storage and processing capability than currently available. Smart information able to advertise its presence on arrival at a node, and able to respond to service availability responses to configure itself for the local environment, will also be available by this date.

10. Conclusions

We have presented an outline design for an active information network environment, and discussed some of the key management, monitoring and control functions we have developed to support the environment. The first integrated demo of the environment is expected in August 1999 on the LEANET infrastructure. Key issues for further research have been outlined. Further collaboration with the academic authors is planned in order to carry out the research.

11. References

- [Alan] M. Fry and A. Ghosh "Application Layer Active Networking", Fourth International Workshop on High Performance Protocol Architectures (HIPPARCH '98), June 98, longer version to appear in *Computer Networks and ISDN Systems* 31 (1999).
- [Alex97] Alexander, Shaw, Nettles and Smith "Active Bridging", *Computer Communication Review*, 27, 4 (1997), pp101-111
- [Alex98] D.S.Alexander et al "A secure active network environment architecture", *IEEE Network* 1998
- [Amir] E.Amir, S.McCanne, R.Katz "An active service framework and its application to real time multimedia transcoding", *Proc SIGCOMM '98* (Sept '98) pp178-189
- [Bash] O Bashir, IW Phillips, DJ Parish, JL Adams and T Spencer, "The Management and Processing of Network Performance Information", *BT Technology Journal*, 16(4), October 1998, pp 203-212

- [Bis] J.Biswas et.al “The IEEE P1520 standards initiative for programmable interfaces”, IEEE Comms. Oct 1998 pp64-72
- [Cao] P. Cao, J. Zhang and K. Beach “Active Cache: Caching Dynamic Contents (Objects) on the Web” Proc middleware '98 (Ambleside).
- [Laz] A.Lazar “Programming Telecommunication Networks”, IEEE Network Oct 1997 pp 2-12
- [Lupu97] Lupu E. and Sloman M. (1997b) *A Policy-based Role Object Model*, Proceedings of the 1st IEEE Enterprise Distributed Object Computing Workshop (EDOC'97), Gold Coast, Australia, Oct.97, pp. 36-47.
- [Mar95] I.W.Marshall and M.Bagley, “The Information Services Supermarket - an Information Network Prototype”, BT Tech. Journal, Vol. 13, No.2, Apr 1995.
- [Marr96] Marriott, D. and Sloman M. (1996) *Implementation of a Management Agent for Interpreting Obligation Policy*. Proceedings of the IEEE/IFIP Distributed Systems Operations and Management Workshop (DSOM' 96), L'Aquila (Italy), Oct. 1996.
- [Marr97] Marriott, D. (1997) *Management Policy for Distributed Systems*, Ph.D. Dissertation, Imperial College, Department of Computing, London, UK, July 1997.
- [Mic95] Microsoft Corp. and Digital Equipment Corp., “The Component Object Model Specification”, Draft Version 0.9, Microsoft Developers Network CD-ROM, Oct 1995.
- [Nah95] K.Nahrstedt and J.M.Smith, “The QoS Broker”, IEEE Multimedia, Vol. 2, No.1, p.53-67, Spring 1995.
- [OMG91] “The Common Object Request Broker : Architecture and Specification”, Object Management Group, OMG Doc. 91.12.1, <http://www.omg.org>, Dec 1995.
- [OMG95] The Object Management Group, *The Common Object Request Broker: Architecture and Specification, Version 2.0*. The Object Management Group, OMG Headquarters, 492 Old Connecticut Path, Framington, MA 01701, USA. July 1995.
- [Paru] G.Parulkar et.al “Active Network Node Project” Washington University St Louis
- [PC97] N. Pryce and S. Crane. *Component Interaction in Concurrent and Distributed Systems*. In Proc. Fourth International Conference on Configurable Distributed Systems, Anapolis, MD, USA. May 4-6 1998. IEEE Computer Society. ISBN: 0-8186-8451-8.
- [Phil] IW Phillips, DJ Parish and T Rodgers, "SMDS Network Performance Measurement", IEE Colloquium on "Practical Experience with SMDS", IEE, London, October 1995, pp 7.1-7.4
- [Rog97] D. Rogerson. *Inside COM – Microsoft's Component Object Model*. Microsoft Press, 1997.
- [Roo] S.Rooney et.al “Tempest: A framework for safe programmable networks”, IEEE Comms Oct 1998 pp42-53
- [Slo94] Sloman, M. (1994). *Policy Driven Management for Distributed Systems*. Plenum Press Journal of Network and Systems Management, 2 (4):333–360, Plenum Press.
- [Tenn] D. Tennenhouse, D.Wetherall “Towards an active network architecture”. Computer Communication Review, 26,2 (1996).
- [Wak] I.Wakeman et.al. “Designing a Programming Language for Active Networks” HIPPARCH '98
- [Wadd98] D.G.Waddington and D.Hutchison, “A General Model for QoS Adaptation”, Proceedings of the Sixth International Workshop on Quality of Service (IwQoS'98), Napa, California, USA, May 1998.
- [Wess] Duane Wessels “Configuring Hierarchical Squid Caches”, AUUG'97, Brisbane, Australia.
- [XML] Extensible Markup Language (XML) W3C Recommendation 10-February-1998
<http://www.XML.com/aXML/testaXML.htm>.