A Scalable Middleware–Based Active Network Architecture for Next Generation Communications

Carl Cook Paddy Krishnan Krzysztof Pawlikowski
Department of Computer Science

Harsha Sirisena
Department of Electrical and Electronic Engineering

University of Canterbury
Christchurch, New Zealand

TR–COSC 03/01, April 2001

The contents of this work reflect the views of the authors who are responsible for the facts and accuracy of the data presented. Responsibility for the application of the material to specific cases, however, lies with any user of the report and no responsibility in such cases will be attributed to the author or to the University of Canterbury.

This technical report contains a research paper, development report, or tutorial article which has been submitted for publication in a journal or for consideration by the commissioning organisation. We ask you to respect the current and future owner of the copyright by keeping copying of this article to the essential minimum. Any requests for further copies should be sent to the author.

Abstract

In conventional data communication networks, the basic network components such as workstations and routers are passive, in which routing decisions are made solely on the basis of the packet header information. In contrast, active networks allow the specification of complex processing instructions at all participating intermediate network routers, providing the on–demand installation of complex network services.

As an adaptation of previous active networks, this report introduces an architecture based entirely in middleware, providing an extensive distributed communications framework. The utilisation of middleware services resolve security, optimisation, and memory-management issues otherwise assumed as inherent, and enables a simple yet highly functional multiple–language interface for the deployment of dynamic protocols. After describing the architectural design, an empirical system evaluation is presented with comparisons to both conventional network protocols, and a well–known existing active network architecture. Results indicate performance improvements over the existing architecture, and identify scenarios where active networks can out–perform the capabilities of their conventional counterparts.

Keywords:

Active networks, distributed Com, dynamic protocols, IP routing, middleware

1 Introduction

As the information available to conventional IP–based routers is confined to fields within packet headers, only limited processing can occur, thus restricting functionality and network capabilities. Whilst some end-to-end services can be provided for within the existing IP infrastructure of simple packet forwarding, the current surge in demand for network-level functionality (as reflected by IETF activities on RSVP and IPv6) suggest that the homogeneous model of IP may no longer be suitable for today's networks [14].

The new active network paradigm, by contrast, provides networked applications and users with programmable interfaces that support the dynamic modification of a networks behaviour, bypassing both the standards committee and the hardware vendor in order to cater for ever-changing user demands. Such flexibility is achieved by allowing the specification and invocation of complex processing instructions at all participating intermediate active network routers, facilitating the run time installation of arbitrary software–based routing protocols [18]. Numerous applications of active networks may be envisioned, such as dynamic compression, arbitrary network caching, and on–demand encryption between endpoints in conventional, multi–casted, and future IP–based networks.

As an adaptation of previous active networks, this report presents an architecture that utilises the services of middleware to facilitate all system tasks. This is an alternative design methodology to the mainstream approach where core active network routines are implemented internally, such as distributed object communication, user authentication, and library loading. By the virtue of middleware, the ComAN architecture gains additional benefits such as language independence and cross–platform interoperability.

Section 2 provides an overview of recent work related to active networks and middleware frameworks, and presents the existing architectures that incorporate middleware–based components. Section 3 discusses the concept of middleware–based active networks. Section 4 presents design and implementation details of the ComAN active network architecture, with an overview of typical system usage. Section 5 details an empirical evaluation of ComAN, with performance measurements contrasted to both conventional network protocols and a functionally–similar active network implementation. Finally, Section 6 discusses various aspects of system performance, flexibility, and security, providing direction for further work.

2 Background and Related Work

2.1 Active Network Overview

Numerous active network architectures and components are currently being investigated by established research groups, with the collective goal to develop systems that address the key aspects of active networks, namely performance, flexibility, and safety [13]. Whilst the concept of active networks is relatively new, several active network implementations have already been released for comment and further development. One of the earliest active networks, ANTS, provides a JAVA–based framework for the development of dynamic network protocols [19]. Many protocols have been successfully emulated using ANTS, such as multicasting [4], mobile–host support [12], and congestion control [3]. SNAPD, a more recent active packet system, demonstrates that user–defined packet–processing routines can be invoked without a detrimental impact on performance [14]. Another recent active network architecture, comprised of BOWMAN and CANEs, demonstrates the feasibility of support for existing protocols over active networks, with results indicating improvements in end–to–end performance [10].

Due to the highly–configurable nature of active networks, new threats are introduced in terms of network safety and security. Accordingly, considerable research effort is aimed towards the establishment of mechanisms to provide network–level safety. SANE provides a layered environment for the construction of secure active network infrastructures [1]. Based on a minimal set of trust assumptions and rules, the environment securely bootstraps an embedded active network,

providing authentication and naming services for active packets. As an alternative approach, the ActiveSpec framework formally verifies the security policies for active networks and their services [6], guaranteeing the integrity of an active network architecture. Through a formally defined set of services, policies, and resources, the interactions within an active network can be modelled with subsequent verification of security requirements.

Recent research has also displayed the potential of active network specific packet–processing languages to address issues of performance, safety, and portability. Such languages attempt to make the execution of arbitrary user instructions on active nodes safer by restricting dangerous constructs of conventional languages, and efficient by reducing the number of primitive operations and tailoring such operations for interpretation optimisations. Plan, a lightweight language based on lambda calculus, acts as a replacement to packet headers in a conventional network [9]. As the successor to Plan, the Snap packet language has been tuned to permit a fast interpretation by active nodes, whilst maintaining safety assertions made possible by a simple grammar [15]. Similarly, as a component of the Smart Packets project, the Spanner assembly–like language provides a simple packet programming environment, with tight restrictions on memory access to strengthen the level of security [17].

2.2 Middleware Overview

With the continued growth in network–based applications, distributed middleware systems (or object request brokers) of various incarnations have emerged to manage issues of complexity and scalability. Common middleware systems include Corba and DCom, with both systems available on Unix and Windows platforms. Introduced by the Object Management Group, Corba is a well specified distributed object protocol that is implemented by various independent vendors, whereas Microsoft's DCom represents a single entity consisting of a wire–protocol and a proprietary implementation. Java's Remote Method Invocation Api is also gaining popularity as a web–based middleware system, with its system independence eliminating the requirement for operating system specific implementations.

The theme of managed distributed object creation, remote method invocation, object persistence, and object termination is common to all distributed middleware systems. Accordingly, the application of middleware is suitable wherever complex distributed functionality is required. Middleware–based systems are commonly utilised when using distributed processing to increase performance, or when implementing redundant links to increase system reliability. From a design perspective, advantages of middleware systems include the separation of interface and implementation [8], support for objects with multiple interfaces, and location transparency. In the case of DCom, language neutrality is also supported natively.

2.3 Existing Middleware–Based Distributed Frameworks

Several middleware–based frameworks have been developed to increase the level of support for next–generation distributed applications. Whilst distributed middleware systems are well suited to conventional client/server based applications, enhanced frameworks have been introduced to provide middleware–specific performance optimisations and QoS features [16]. One such framework, Tao, is a real–time object request broker which provides high–performance middleware components for common networking tasks, with multiple–layer QoS support. Tao has recently been used to implement a framework which supports dynamically–specified transport protocols to support low latency, high bandwidth data streaming. Through the flexibility of its middleware composition, optimised scheduling and buffering techniques were utilised to provide low latency, high throughput video/audio links, suggesting that middleware is highly suitable for next generation distributed applications [11].

A similar framework for middleware–based QoS support has been proposed by Becker et al. [2], which includes a Java–based prototype implemented within a gigabyte router testbed. Another Java–based distributed architecture, Chameleon, also focuses on middleware support for enhanced QoS services, in the domain of high–latency environments such as the Internet [5]. As the

goal of the architecture is to provide a highly–adaptable service, a reconfigurable protocol stack to support arbitrary user–defined protocols has been implemented. The architecture also provides application–level control over common transmission and synchronisation tasks.

3 Merging Middleware and Active Network Concepts

A close relationship exists between distributed middleware systems and active networks. Active networks aim to provide efficient, complete, and highly flexible network infrastructures, and distributed middleware systems appear capable of enabling such systems, either in part or in whole. Despite this close relationship, the mainstream approach to active network design appears to involve the internal implementation of mechanisms for distributed communication, library loading, remote method invocation, and security management, with little consideration for existing frameworks that support such tasks. The architecture presented in this report presents a case for active networks that are based entirely in middleware, avoiding the need to "reinvent the wheel" at both the architectural and implementation levels.

Although active network frameworks provide a great degree of application–level flexibility, for the most part they are closed systems that do not provide services for existing legacy client/server applications. By utilising middleware support, existing distributed applications may be easily modified to connect to active network architectures. In a similar manner, the presence of well–defined middleware interfaces allows the bridging of different active network architectures, forming potentially global active networks without the necessity for an intermediate encapsulation protocol. As bindings exist to allow native communication between DCom, Corba, and Java, truly global open system communication between middleware–based active networks is possible, regardless of the implementation specifics.

Many other benefits are gained from the use of middleware–based active networks. After the establishment of middleware interfaces for packet processing routines, systems to support user–defined protocols within middleware–based active networks may be simplified. This is achieved by implementing user–defined packet processing routines as middleware components, which are then serialised, transported, instantiated, and invoked automatically by the middleware services of the host architecture. The component developer has the freedom to implement such packet processing components as desired, and end applications or users can request the services of such routines through standard middleware conventions, without any knowledge of the component's implementation details.

Middleware systems also offer extensive operating system support for authentication of users and processes. A fine level of control over the registration, activation, invocation, and termination of packet processing routines is provided, a service that is core to any secure active network architecture. Additionally, the service of user process isolation is also common to middleware architectures, securing middleware–based active network architectures from client crashes. Memory protection is also provided by default, implying that client processes can only access data within their own address space. Any access to other data such as router state information must authenticated, and invoked through well–defined interfaces.

As middleware components are applications in their own right, components loaded and invoked as plugin protocols from within middleware–based active networks have no limits on service possibilities. Middleware components contain their own data structures, and may request the same active network services that are available to external applications. As an example, packet processing components can query the active network routers for the current rate of throughput, and reroute packets according to QoS constraints. Intelligent caching, multicasting, congestion/admission control, mobile host support, Nack implosion protection, compression, and encryption are just a few of the many protocols that can be readily implemented as middleware components. Such components, when registered, can be utilised by an application throughout the network with a single method call.

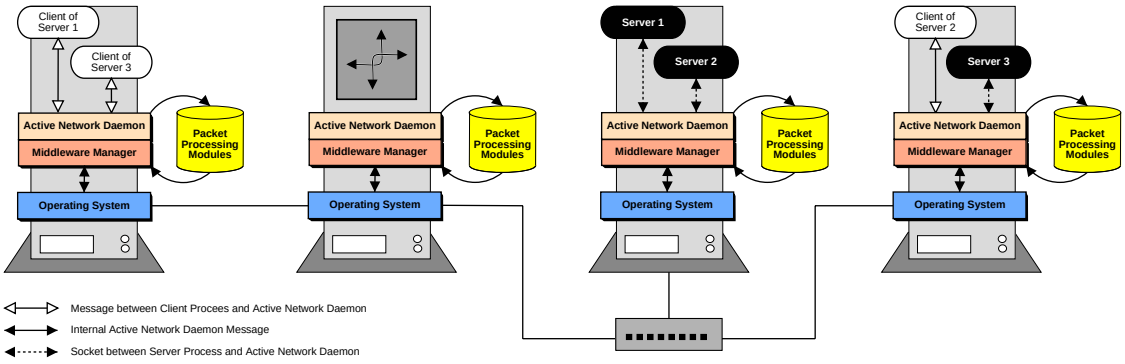


Figure 1: The high–level system architecture of ComAN.

3.1 Existing Middleware–Based Active Networks

Several existing systems can be identified as middleware–based active networks. As a three year collaborative research project, Fain represents an extensive middleware–based active network architecture, aiming to provide a highly configurable integrated hardware/software system [7]. The Fain enterprise model has now been specified, with current work focusing on the development of an active node platform layer, a service programming environment, and a management system.

Whilst not cited as active networks, the middleware–based Chameleon and Tao frameworks also offer services that offer similar functionality (see Section 2.3). Regardless of their intended application, the findings of these systems provide valuable insight for the development of dedicated middleware–based active networks. Finally, the Ants active network can also be viewed as a middleware–based, due to the inherent use of Java's distributed object communication model. However, Ants does not yet provide for open system communication and multiple–vendor support.

4 ComAN: A Middleware–Based Active Network

ComAN is designed as a lightweight yet practical desktop–to–desktop active network architecture, readily installable throughout a network of any topology, providing flexible packet processing and routing for next–generation services and applications. A key feature of the architecture's design is the high level of connectivity available to networked applications that utilise ComAN's services, and to other middleware–based active networks. ComAN is also designed to enforce strong levels of user and process authentication.

The high–level architecture of ComAN is presented in Figure 1, illustrating that socket–based client/server streams may connect through the ComAN service resident at each *active node*. By default, ComAN forwards all data packets to the requested destination without payload modification. However, packets may be subjected to additional processing and rerouting upon request, through the specification of user–defined packet processing routines, or *router modules*.

4.1 Service Establishment

After attachment to the ComAN service, a client may specify a destination node where a conventional socket–based server process is listening. This commences the establishment of an *active channel*. At the source node, ComAN uses Icmp messaging to determine the next physical hop to the destination, based on the underlying route table. ComAN will then attempt to attach to the ComAN service running on the next hop, forming the first logical link in the active channel. This process continues until the destination node has been reached, at which point a local socket connection to the server process is created, and the active channel is established.

IComan		
AddManditoryModule	Send	GetModuleData
RemoveManditoryModule	SendTo	GetCurrentBPS
AddModuleToStream	SendUsingModule	CloseSocket
RemoveModuleFromStream	Recv	
CreateSocket	RecvFrom	
ConnectSocket	RecvUsingModule	

IRouterModule
GetReRoutingMode
GetPacketProcessingMode
GetBlockingMode
GetData
ReRoutePacket
ProcessPacket

Table 1: The core API for ComAN.

Upon establishment of an active channel, the client may begin sending and receiving data. When data is sent, it is placed into the local packet queue. The ComAN *worker thread* processes and routes each packet according to the instructions of the router modules associated with the channel. If a packet is not rerouted by any of the channel's router modules, the packet is forwarded to the next active node in channel's path. This process continues until the destination node is reached, at which point the data is delivered to the server process through the local socket. Similarly, each active channel has a dedicated *listener thread* that is bound to the server process at the destination node. Upon receipt of data from the server socket, the listener thread places the data in the channel's received packet queue. This data may be removed from the queue by the client through a blocking method call, emulating the Bsd `recv` socket routine.

4.2 Architectural Features

The following paragraphs expand upon ComAN's key features, providing more details of the active network's architectural design:

Flexibility Router modules may be developed in many languages, including C, C++, Java, Cobol, assembly language, and Pascal. Such modules are implemented as middleware components that adhere to the common packet routing interface (see Section 4.2), providing a simple "plug–and–play" mechanism for both module development and invocation. No bounds are placed on the functionality of such modules, and data streams utilise the services of these modules through a single method call. ComAN also provides control over the invocation of router modules. All network packets can be subjected to any given router module, or invocation of modules may be restricted to all packets in a single stream, or to individual packets. Packets may be subjected to processing from the sender to the receiver, the receiver to the sender, or in both directions.

Practicality ComAN executes on an unmodified operating system, entirely in user address space. No additional drivers are required, and the installation procedure is minimal, requiring no system restarts. Similarly, packet processing modules may be installed and deinstalled upon request, without interruption of the active network service. Additionally, it is not essential that ComAN resides on every network node. In cases where the active network service can not be installed, for example at a gateway router, ComAN will revert to IP packet forwarding across the node. Subsequently, not even the endpoint nodes are required to have ComAN installed, although the existence of the active network service at endpoints will be beneficial in most cases.

Packet forwarding is based on operating system route tables, which allows the system to adapt instantly to changes in the underlying route tables. For simplicity, ComAN uses conventional IP forwarding logic when routing packets, unless user–defined routing is requested. ComAN also supports any number of concurrent connections, providing a full forwarding service where packets are placed into a Fifo queue, processed, and then routed. Additionally, ComAN offers a simple API which emulates the Bsd socket routines for the forwarding of data, with extra mechanisms to specify the invocation of router modules. The explicit formation of packets and marshaling of data is not required when using ComAN, as such tasks are implemented internally.

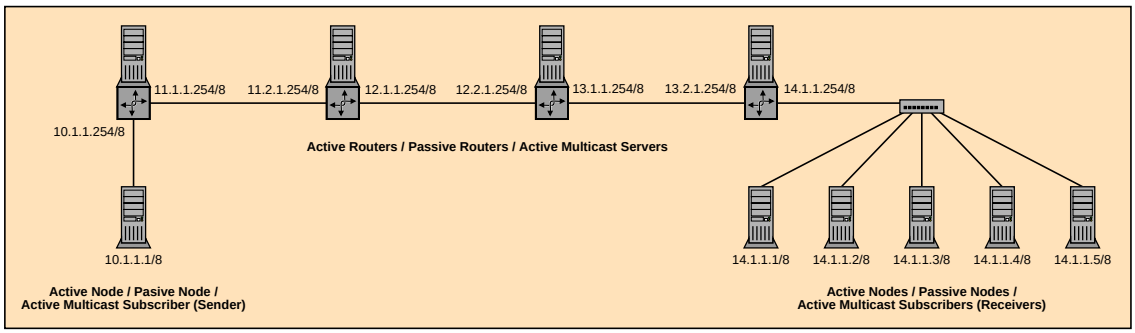


Figure 2: The network topology and link–layer configuration for evaluation of ComAN.

Security As a service of DCom, authentication of all active network components is maintained by the operating system, at both user and machine level. Subsequently, it is possible to restrict which users or groups of users may connect to ComAN and register, load, invoke, and unload router modules.

Accessibility DCom bindings exist for C, C++, Java, and Vb, allowing multiple–language access for clients of the ComAN active network. Additionally, several remote instances of ComAN may communicate natively to form one logical active network, with the architecture reverting to conventional IP forwarding for all passive intermediate nodes. Similarly, other middleware–based active networks have the potential to communicate directly with ComAN, due to the middleware bindings that exist between DCom, Corba, and Java.

4.3 The Network Api

As presented in Table 1, the Api for ComAN consists of two main interfaces. The `IComan` interface is returned to clients upon attachment to the ComAN service. This interface emulates conventional socket routines, provides support for the registration of router modules, and allows the retrieval of node and router module information. As an example of typical usage, the following simple C–based file transfer routine demonstrates the creation of an active channel, the registration of a router module, the connection of the server socket, and the transmission of user data:

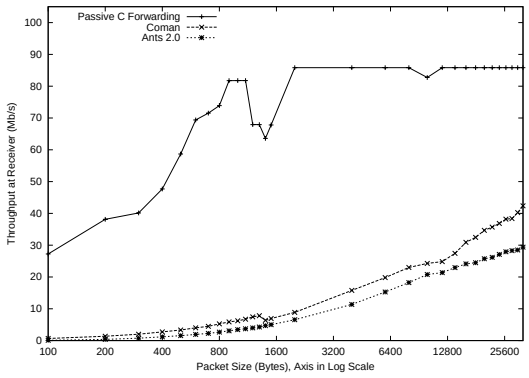
```
p_IComan–>CreateSocket(AF_INET, SOCK_STREAM, "Server4", SENDPORT, &SvrSocket);
p_IComan–>AddModuleToStream(SvrSocket, CLSID_ENCRYPT_MOD);
p_IComan–>ConnectSocket(SvrSocket, TRUE);
while(fread(Buffer, 1, BUFLEN, p_File)) { p_IComan–>Send(SvrSocket, Buffer, BUFLEN); }
p_IComan–>CloseSocket(SvrSocket);
```

The `IRouterModule` interface specifies the routines that all ComAN–compliant router modules must implement. This interface is used internally by ComAN when routing packets through the network. Through the appropriate interface routines, ComAN determines which modules intend to reroute and/or process packets, and invokes the processing and routing methods accordingly. Implementation of such routines is module–specific.

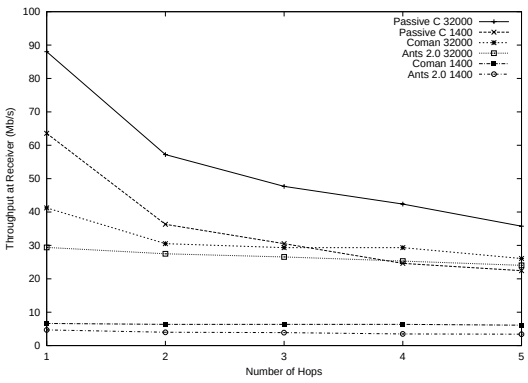
5 Architectural Evaluation

5.1 Experimental Design

The performance of ComAN in terms of latency and throughput was measured on a switched Ethernet network of six workstations and four routers, with the topology presented in Figure 2. Each workstation and router had a PC clone architecture, an ASUS 33 MHz single Cpu mainboard,



(a) The average throughput of single–hop sustained transmission using packets of up to 32000 bytes.



(b) The average throughput of multiple hop sustained transmission with fixed packet sizes.

Figure 3: The performance of C forwarding, ComAN, and Ants in terms of throughput.

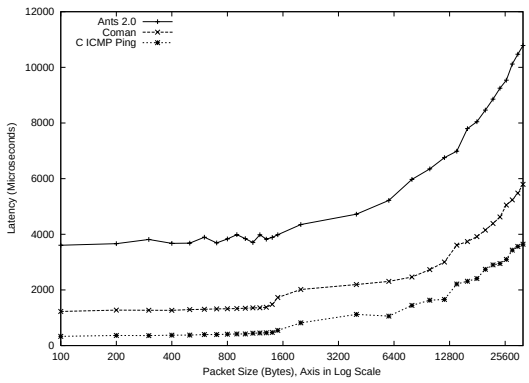
128 Mbs of Ram, and an Intel Pentium II Cpu with 16 Kb on–board cache memory operating at 233 MHz. All network cards were Intel Pro 100+B's running at 100 Mbps in full duplex mode, and the operating system used throughout the network was Windows 2000 Advanced Server, build 2195, service release 1. The routers in the network consisted of multihomed workstations with static routes, with each router bridged by crossover Ethernet cables. All broadcast and background traffic was eliminated prior to testing. Finally, as Ants was also tested on the network, Sun Microsystems' release of Jdk 1.2.2 used for compilation of the Ants toolkit and related test applications, and the Java 1.2.2-001 native threads virtual machine used for interpretation.

5.2 Results

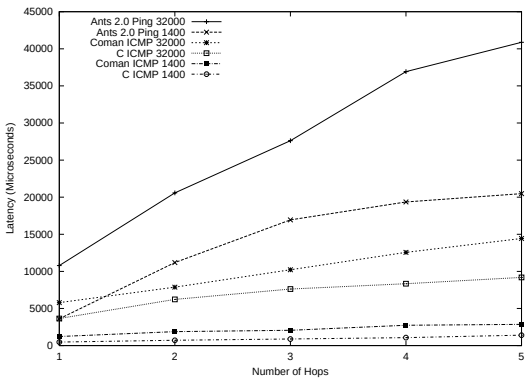
Throughput The sustained end–to–end transfer of consecutive Null characters was used to measure throughput rates. As a test mechanism, a Tcp–based server process was bound at the destination node, where the server process would continuously accept data immediately after the establishment of a client connection. To determine the throughput rate achieved, ComAN established an active channel from the source node to the server process, and commenced the transmission of 200 Mbs of data. The time taken to transmit the data was then used to calculate the transfer rate, which was then averaged over 30 trials. As comparative measurements, a passive C–based client and the Ants active network architecture were also tested under the same conditions. The Ants application developed to measure rates of throughput was translated as closely as possible from the relatively generic C and ComAN–based client/server programs.

The effect of packet size on the rate of throughput over a single hop route is presented in Figure 3(a). This measurement provides an indication of the baseline throughput capabilities, without consideration of any active processing overheads that may be incurred by the active network architectures. The results show that C–based forwarding could achieve a throughput of 82 Mbps with a packet size as small as 1000 bytes, yet even with a 32000 byte packet size, ComAN achieved only 42 Mbps, followed by Ants with 29 Mbps. Whilst only slight variance was experienced, all methods were subjected to performance degradation when packet sizes approached 1500 bytes, due to link–layer fragmentation.

The effect of multiple hop routes on the rate of throughput is presented in Figure 3(b). All three data transfer methods tested displayed varying degrees of performance degradation as the number of hops from sender to receiver increased. Whilst C forwarding appeared to be most affected by multiple–hop data transfer, the forwarding capability of the network was the actual limiting



(a) The average latency of single–hop ping with packets of up to 32000 bytes.



(b) The average latency of multiple–hop ping, with fixed packet sizes.

Figure 4: The performance of C forwarding, ComAN, and Ants in terms of latency.

factor. The routers in the network are software based, and the hardware is of a relatively low specification, which subjects C forwarding to confounded performance degradation as it reaches the throughput threshold for the network. The throughput rates achieved by the C forwarding did however provide an indication of the upper bound for the network's routing capacity.

The rate of throughput achieved by ComAN reduced from 42 Mbps for single hop data transfers to 30 Mbps for multiple hop data transfers using 32000 byte packets. The 12 Mbps degradation in performance for multiple–hop data transfer represented the additional delay incurred by the active nodes that routed packets onwards, where packets were both delivered to and sent from the user–level ComAN service. An additional decrease in performance was experience for five–hop data transfer using ComAN with 32000 byte packets, however, again this is likely to be due to the limitations of the network rather than a performance issue with ComAN. When reduced to 1400 byte packets, both ComAN and Ants demonstrated poor rates of throughput, with approximately an 85% deterioration in performance when compared to the throughput rates obtained using 32000 byte packets. C forwarding's rate of throughput decreased on average by 40% when 1400 byte packets were used.

Latency To measure the latency of ComAN, a ping application was implemented. As comparative measurements, ping applications written in C and Ants were also developed. The C–based version used Icmp messages, and for Ants, a ping application was readily available from the Ants 2.0 distribution. However, as the original Ants version of ping had only millisecond granularity, the program was modified to call the high–precision timer as used by the C and ComAN applications. Identical to the throughput trials, latency was tested in terms of both packet size and number of hops between endpoints. The response times as presented in Figure 4 represent the average of 200 consecutive pings, where times for each round–trip ping were halved on the basis that all delays within the network were symmetrical.

As presented in Figure 4(a), all ping operations over a single–hop route displayed a linear increase in latency as the packet size increased. C–ping had a relatively low latency (ranging from $300\mu s$ to $3600\mu s$), with ComAN–ping displaying approximately $1100\mu s$ more latency regardless of the packet size used. Ants–ping was considerably more latent, at least doubling the latency of ComAN–ping for every packet size trialed.

Each version of ping displayed an increased latency according to the number of hops between endpoints, as presented in Figure 4(b). When using 1400 byte packets, both C–ping and ComAN–ping doubled their latency when comparing response times from a single hop to a five

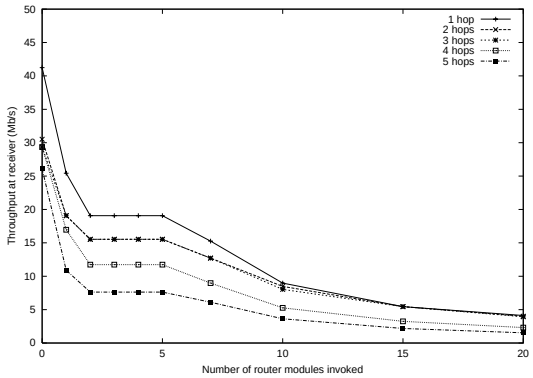


Figure 5: The impact of ComAN router module invocation on throughput rates.

hop route. At five hops, C–ping averaged 1400μs, with ComAN–ping averaging 2800μs. However, Ants–ping had increased its single hop latency by a factor of five, with an average five hop latency of 20500μs. Using 32000 byte packets, each version of ping approximately tripled its latency for five hops when compared to a single hop route. For a five hop ping, C–ping averaged 9000μs, ComAN–ping 14000μs, and Ants–ping 41000μs.

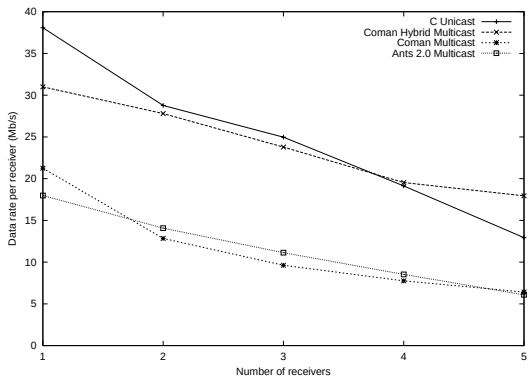
Impact of Router Module Invocation To measure the impact of ComAN router module invocation on the rate of throughput, multiple router modules per node were invoked on an active channel. The active channel was dedicated to the sustained transmission of client data, using the same throughput process as outlined previously. Each router module simply set the first byte in each packet from a Null to EOF, providing an unbiased measurement of the overhead related to the invocation of router modules, regardless of additional user–defined processing instructions. Rates of throughput were measured for routes of up to five hops from sender to receiver.

As Figure 5 presents, the invocation of a single module reduced throughput by up to 50%, regardless of the number of hops in the route. For the invocation of between 2 to 5 modules, throughput remained constant but at a deteriorated rate. As expected, deterioration in throughput was greater as the number of hops in the route increased, due to the active processing performed at each hop. For example, each packet of a single hop / five module active channel are processed 10 times in total, whereas a five hop / five module active channel processes each packet 30 times prior to application delivery. As the results show, throughput rates continued to deteriorate as additional modules were loaded.

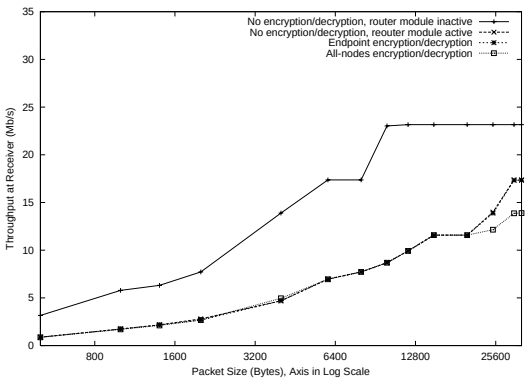
5.3 Architectural Application

Multicasting A typical application of active networks is the emulation of multicasting services where no native multicast support exists. Accordingly, the throughput rates for ComAN and Ants multicasting were evaluated, with results presented in Figure 6(a). The evaluation identified a scenario where active networks provide throughput gains over conventional unicasting by reducing the number of redundant packet transmissions. The network topology used in the evaluation is presented in Figure 2. In all experiments, the sender (10.1.1.1) transmits multicast packets through four active routers, arriving at up to five receivers on the end subnet (14.1.1.*). The data transfer method used was identical to the sustained throughput method as outlined previously, with 32000 byte packets.

Four applications for the transfer of multiple data streams were trialed. A C–based unicast application was developed to provide standard unicast streaming. Multicast emulation applications were developed using ComAN and Ants, where the exiting Ants multicast application from the Ants 2.0 distribution was modified to enable sustained data transfer. Finally, a hybrid application



(a) The throughput rates for unicast and emulated multicast data streaming, using 32000 byte packets.



(b) Impact on throughput for five–hop encryption, using packets of up to 32000 bytes.

Figure 6: The application of the ComAN architecture in terms of throughput

was developed using ComAN, aimed to provide improved throughput rates. The application established an active channel through the intermediate routers, but reverted to conventional unicast transfer between the end router and the receivers. This technique disallows any active processing at the receiver nodes, but vastly reduces the active processing requirements at the end router.

As Figure 2 presents, the C–based unicast application achieved throughput rates that were approximately twice as fast as the rates achieved by the ComAN and Ants multicast applications, regardless of the number of receivers. However, for four or more receivers, the hybrid ComAN application outperformed the C–based unicast application, providing at least 18 Mbs of data per second to each receiver. With five receivers, the burden of sending duplicate packets over the intermediate routers restricted the C–based unicast application to client throughput rates of 13 Mbps. Additionally, it is likely that with a high–performance server architecture for the heavily loaded end router, ComAN's end–to–end performance would improve considerably, allowing a full end–to–end active multicast service with throughput rates similar to that of the hybrid version.

Encryption User–implemented encryption of data streams is a practical application of ComAN. To evaluate the impact of user–defined packet processing routines on throughput rates, an encryption router module was implemented. The router module provided X–Or bitwise encryption with a known key, where each byte in every packet was encrypted. Decryption consisted of applying the encryption step again with the same key. An active channel was established to transfer a sustained stream of data to a listening Tcp–based server process at a destination node, and the encryption module was invoked on all packets in this channel. A five hop route was tested, with packet sizes ranging from 100 to 32000 bytes.

To provide insight as to where the router module processing overhead occur, various levels of module invocation were tested over distinct data streams. As a baseline comparison, data was transmitted without any router module invocation. Data was also transmitted with a router module that did not perform any processing, in order to measure the overhead of per–packet router module communication. Next, data was transmitted with a module that would encrypt all packets at the source node, decrypt all packets at the end node, and inspect yet leave packets unmodified at all intermediate nodes. Finally, data was also transmitted with a module that also performed decryption and re-encryption of data at all four intermediate nodes.

As presented in Figure 6(b), user–defined processing had little direct influence on the deterioration of throughput rates, unless packet sizes were over 20000 bytes. Rather, the invocation of the router modules, regardless of their user–defined routines, was the main cause of performance

degradation. On invocation of a router module, each channel experienced approximately a 50% loss in throughput for most packet sizes, regardless of the level of user–defined processing specified by the router module.

5.4 Observations

During all trials, a positive linear correlation was present between the rate of throughput and packet size for both ComAN and Ants. When large packet sizes are used, fewer active processing steps per byte are required. Such packets are fragmented at the network layer and routed as a sequence of smaller sized packets at the link layer, which is similar to the technique employed by tag switching, where processing overhead per byte is reduced by diverting network layer–routing to link–layer switching after the identification of a packet stream.

Using low–specification desktop hardware, both ComAN and Ants failed to provide rates of throughput capable of saturating a 100 Mbps Ethernet link. This implies that neither of the two active network architectures are currently suitable for use in an environment where high speed transmission is required, unless high–performance hardware is used. However, ComAN did provide relatively low latency, even with large packet sizes, indicating that the architecture is well suited to delay–sensitive applications, such as realtime media streaming.

As indicated by the evaluation of throughput rates, ComAN's processing bottlenecks occurred at the active network routers. After profiling the active routers under a heavy loading, it was revealed that the number of Cpu cycles consumed when copying data from kernel to user address space, and the time spent waiting for I/O routines to complete, were the main factors limiting ComAN's performance. By applying more processing power to active network routers in the form of multiple Cpu machines and server architectures designed for high I/O utilisation, the performance threshold of ComAN is expected to increase throughout the network.

6 Discussion and Further Work

The multicasting and encryption modules presented during the trial of ComAN demonstrate the architecture's flexibility, performance, and practicality. The hybrid multicasting module provided a greater level of end–to–end throughput than conventional IP–based streaming for the congested network topology, and the encryption module allowed a different encryption realm over each hop in the network, with an acceptable decrease in throughput. Both of these modules may be freely distributed to active networks where such services are required, with the functionality accessible through a single call to the user Api.

Whilst ComAN stands as a useful prototype, an important next step is the development of a standard set of middleware interfaces, allowing integration with other middleware–based active networks such as Fain. Additionally, as middleware systems do not support code verification, the development of an enhanced middleware framework supporting code verification is also to be investigated. To increase safety and performance, ComAN, with the services of security–enhanced middleware, has the potential to support the Snap packet language among others, providing an interoperable, efficient, and secure active network architecture.

Other developments of ComAN include porting the architecture to Unix. This is expected to be a trivial task, but if difficulties arise, the migration of the architecture to Corba on both the Windows and Unix platforms will be investigated. Additionally, ComAN currently has rather naïve admission control mechanisms which could be tested and developed further. The development of QoS–specific router modules to assist the management of complex topologies is also envisaged.

7 Conclusions

Middleware systems provide many useful services for the both the development of active networks, and the integration of active networks with their host applications. By utilising such services,

ComAN provides a practical yet lightweight active network architecture for the deployment of language–independent dynamic protocols. Additionally, the ComAN active network architecture is readily installable throughout a workstation–based network, providing operating system supported user authentication, and reasonable protection from malicious or erroneous system use.

Using the concept of pluggable middleware components, new protocols and services can be installed at runtime. Scenarios have been identified where through the use of such components, ComAN improved the quality of application–based data transfers in terms of both performance and flexibility. Whilst evaluation showed that raw data transfer rates are slower than the rates achieved by conventional networks, more powerful router hardware should address the outstanding performance issues.

As the concept of a functional middleware–based active network has been proved, attention now focuses on integration with existing architectures for increased flexibility, safety, and performance.

References

[1] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. Safety and Security of Programmable Network Infrastructures. In *Communications Magazine*, pp. 84 – 92. IEEE, October 1998.

[2] T. Becker, H. Guo, and S. Karnouskos. Enable QoS for Distributed Object Applications by ORB–Based Active Networking. In *The 2nd International Conference on Active Networks (IWAN)*, edited by H. Yasuda, Volume 1942, pp. 225 – 238. Lecture Notes in Computer Science Series, Tokjo, Japan, October 2000.

[3] S. Bhattacharjee. On Active Networking and Congestion. Technical Report GIT-CC-96/02, Georgia Institure of Technology, February 1996.

[4] M. Calderon, M. Sedano, A. Azcorra, and C. Alonso. Active Network Support for Multicast Applications. In *Network Magazine*, pp. 46 – 52. IEEE, May 1998.

[5] K. Curran and G. Parr. A Reflective Framework for Mixed Data–Streams on the Internet, February 2001. Distributed Multimedia Research Group, University of Ulster.

[6] D. Dieckman, P. Alexander, and P. A. Wilsey. ActiveSpec: A Framework for the Specification and Verification of Active Network Services and Security Policies. In *Thirteenth IEEE Annual Symposium on Logic in Computer Science*. IEEE, Indianapolis, Indiana, 1998.

[7] A. Galis, B. Plattner, E. Moeller, J. Laarhuis, S. Denazis, H. Guo, C. Klein, J. Serrat, G. T. Karetsos, and C. Todd. A Flexible IP Active Networks Architecture. In *The Second International Conference on Active Networks (IWAN)*, edited by H. Yasuda, Volume 1942. IEEE, Springer Press, Tokjo Japan, October 2000.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object–Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995.

[9] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM International Conference on Functional Programming Languages (SIGPLAN)*, pp. 86–93. ACM, 1998.

[10] M. Keaton, S. Bhattacharjee, M. Sanders, K. Calvert, E. Zegura, and S. Zabele. Active Reliable Multicast on CANEs: A Case Study. In *Proceedings of the 4th International Conference on Open Architectures and Network Programming (OPENARCH)*. IEEE, April 2001.

[11] F. Kuhns, C. O'Ryan, D. C. Schmidt, O. Othman, and J. Parsons. The Design and Performance of a Pluggable Protocols Framework for Object Request Broker Middleware. In *Proceedings of the IFIP 6 th International Workshop on Protocols For High-Speed Networks (PfHSN '99)*. IFIP, Salem, MA, August 1999.

[12] U. Legedza, D. Wetherall, and H. Guttag. Improving the Performance of Distributed Applications Using Active Networks. In *Proceedings of the 17th Conference on Computer Communications (INFOCOM)*. IEEE, San Francisco, California, April 1998.

[13] D. Maughan. Active Network Research Web Site of the Darpa Information Technology Office. Url, September 1997. `http://www.darpa.mil/ito/research/anets`

[14] J. T. Moore. Safe and Efficient Active Packets. Technical Report MS-CIS-99-24, Department of Computer and Information Science, University of Pennsylvania, October 1999.

[15] J. T. Moore and S. M. Nettles. Towards Practical Programmable Packets. In *Proceedings of the 20th Conference on Computer Communications (INFOCOM)*. IEEE, Anchorage, Alaska, April 2001.

[16] D. C. Schmidt, D. L. Levine, and S. Mungee. The Design and Performance of Real-Time Object Request Brokers. In *Computer Communications*, Volume 21(4):pp. 294 – 324, April 1998.

[17] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, D. Rockwell, and C. Partridge. Smart Packets for Active Networks. In *ACM Transactions on Computer Systems*, Volume 18(1), February 2000.

[18] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. In *Computer Communication Review*, Volume 26(2), April 1996.

[19] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. Ants: A Toolkit for Building and Dynamically Deploying Network Protocols. In *Proceedings of the 4th International Conference on Open Architectures and Network Programming (OPENARCH)*, pp. 1 – 12. IEEE, San Francisco, California, April 1998.