Active Routers in Action: Evaluation of the LARA++ Active Router Architecture in a Real-Life Network[1]

Tim Chart, Stefan Schmid, Manolis Sifalakis, and Andrew C. Scott

Distributed Multimedia Research Group, Computing Department,
Lancaster University, Lancaster, LA1 4YR
{chart, sschmid, mjs, acs}@comp.lancs.ac.uk

Abstract. The paper reports on lessons learned from developing, deploying and operating LARA++ based active routers [1] along with a number of active services used daily in a real-life network environment. We evaluate how LARA++, which claims to be a truly generic and flexible active router architecture, performs when deployed in an operational network. The architecture is assessed whilst providing a range of diverse active services from well-known network services like NAT and firewalls to novel types of services such as Mobile IPv6 handoff optimisation. A particular challenge we consider is the transparent and concurrent introduction of new services by different end users and network administrators.
Besides a qualitative evaluation of the LARA++ architecture, the paper provides a number of quantitative results that show the performance of LARA++ whilst providing the different services. The results indicate that LARA++ not only supports highly generic programmability by independent users, but also provides sufficient performance for today's edge networks.

1 Introduction

Active and programmable network technologies were conceived to allow the introduction of new, value added services inside the network. This is particularly the case for edge networks, close to the end-user, where the demand for such services is highest. This trend is seen in today's networks, where most sophisticated functionality remains within the access or edge networks (for example, firewalls, NATs and v4/v6 transitioning mechanisms), while the core evolves into a 'simple' high-speed routing/switching network.

To be successful in edge networks, active and programmable network architectures must be sufficiently flexible and generic to accommodate not only a diverse range of current services, but also future services that have not yet been conceived. Moreover, the fact that routers are shared resources, which have to provide network services for end-users with different application and service requirements at the same time, further emphasises the importance of flexible and generic programmability.

[1] This research is funded by the EPSRC (grant number GR/R31461/01)

N. Wakamiya et al. (Eds.): IWAN 2003, LNCS 2982, pp. 215–227, 2004.

Most current active router platforms fall at this first hurdle. Some are built with only a specific range of services in mind (for example, network management [2,3], signalling [4], and multicast [5,6]), while others fail to provide the necessary flexibility or extensibility to dynamically deploy new types of services, which have not been considered by the developers (for example, Scout [7], Click [8] and RouterPlugins [9]). Yet, the most neglected requirement in today's active router architectures is support for service composition enabling the installation and management of diverse services by independent users who may be unaware of each other.

In this paper we analyse the experiences gained from deploying active routers in real edge networks where several network services such as network address translation, firewall and mobile handoff optimisation are needed. We have learned that any active router platform that aspires to be genuinely useful in such an environment must exhibit a number of properties. It must be generic and flexible enough to support a diverse range of active services. It must also allow multiple services to co-exist and, where necessary, overlap so that a suitable service composite is generated. And finally, the service composition framework orchestrating multiple services on an active router must allow independent users (for example, network administrators and end users) to integrate active services in a meaningful and structured manner. The main aim of this paper is to relate our experience of deploying actual services on a real active router platform exhibiting these properties.

The rest of this paper is structured as follows: Section 2 recaps the main elements of the LARA++ architecture that are relevant to this paper. In section 3, we describe three LARA++ active components that have been developed, deployed and tested in a real-life edge network. Section 4 relates LARA++ to other active and programmable network solutions. Finally, we conclude in section 5 with some lessons learnt from our experiences with LARA++.

2 LARA++ Architecture

LARA++ [1] is a software augmentation designed for router and commodity operating systems; implementations currently exist for Microsoft Windows XP, Server 2003 and also Windows CE. LARA++ embodies a framework that exposes a programmable interface, which allows active programs, referred to as *active components*, to provide network services on LARA++ active nodes, and will operate on any packet or frame-based network.

LARA++ installs "hooks" directly into the host OS in order to allow the transparent interception of packets passing through the node. Packets are re-injected back into the host OS subsequent to processing. The placement of these hooks is crucial to the interoperation of LARA++ active components with the conventional network services of the router. Since packets are intercepted and re-injected in such a way that they still pass though the host-provided routing and delivery engine, LARA++ can extend the functionality of the router's conventional network services. Although this is not a required behaviour, as packets can also be fully processed and directly forwarded by LARA++ components, it enables lightweight augmentation of network services and allows for gradual replacement of conventional router functions.

LARA++ treats a router as a resource shared by all of its users. The extent to which it is programmable by individual users is configurable by the router administrator. By

means of node-local policies, the administrator can restrict access to packets that pass through the router and the resources of the router. In addition to the security provided by resource and access control, users are protected from each other's actions by a safety model that gives each component a sandbox called a *processing environment* (PE). LARA++ also allows users that trust each other and all of the components of a user to execute inside the same PE. Because this safety is provided at the operating system level, LARA++ does not need to restrict programmability using language safety features like dynamic type checking or by enforcing the use of type-safe languages such as Java or Caml [10].

LARA++ uses a sophisticated model for service composition [11] enabling components from many sources, including users and applications, to be merged into a single composite service. Each component that is to become part of the service composite installs *packet filters* into the nodes of an extensible directed graph, referred to as the *classification graph*. The classification graph defines the structure and semantics for binding active components together in a meaningful way; in other words, providing the 'glue' for the composition of the overall service.

Packet filters inserted by components into the nodes of the classification graph specify the kind of packets a component wishes to process. Each filter consists of a number of rules that LARA++ uses to determine whether packets match the filter. While these *component filters* hook the functionality or service provided by a component into the desired position on the forwarding path, other filters, referred to as *graph-filters,* are used to form the arcs between nodes in the classification graph. Figure 1 illustrates this in more detail and illustrates how conditional branches in the graph allow packets to be reclassified as they traverse the graph.

This model of service composition offers several major advantages over other active router classification systems. The use of a dynamically configurable and extensible classification graph allows LARA++ to process packets on any type of packet-based network from standard IP networks, through networks with ANEP-based active packet encapsulation [12], to bespoke signalling networks. LARA++ is simply configured with an appropriate classification graph by the node administrator.

In order to make use of the graph, users must know its basic structure. For that reason, we define a well-known structure for general processing of packets in IP and IPv6-based networks. This structure, called the *classification graph table* (CGT), may be customized with finer-grained classification where appropriate.

The filter-based packet interception and the classification graph make LARA++ a platform on which active components with overlapping interests, in terms of the packets they wish to handle, can operate without disrupting each other. For example, if one component installs a filter to express an interest in all packets containing a specific IP option, and another expresses an interest in all TCP SYN packets, the classification graph delivers packets that match both filters to both components in the order defined by the graph. Therefore, service composition on a LARA++ active router is defined implicitly by the classification graph, and it can be exploited as a medium to manage both co-operation and competition between active components.

As packets traverse the classification graph, they are redirected to components with matching filters, where they are processed. Once the component has released the packet, the process of classifying the packet is resumed in the classification graph. Both explicit and implicit ordering of processing can be achieved by placing filters at different points in the graph, and through the structure of the graph, components can co-operate and compete safely.

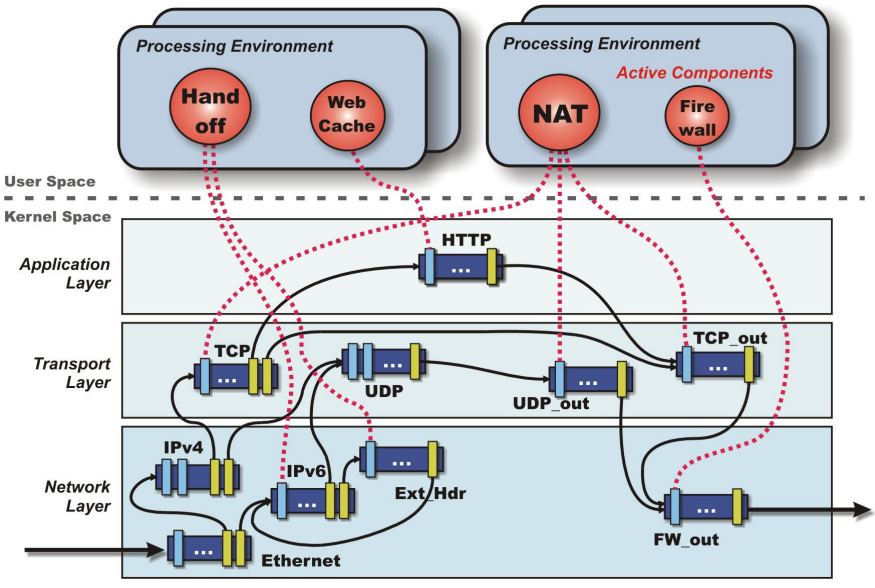

Fig. 1. Classification Graph, showing how components express interest in classes of packet.

Packet filters are extremely flexible from the component writer's viewpoint as they can limit component processing to specific packet flows. Packets can be identified using a powerful bit pattern recognition mechanism and by performing calculations and comparisons on the information in the packet. Once a packet is matched, the filter defines how to direct packets around the classification graph and send packets up to components for processing. The whole system of the classification graph and packet filters is protected from misuse by a permission-based security scheme.

LARA++ filters are easily specified and installed by active components using an XML-based mark-up language supporting the complex nature of filters, which have to be flexible enough to give components the power to identify classes of packets based upon non-trivial features and characteristics. The filter description language provides a similar degree of flexibility to that provided by the Berkeley Packet Filter [13], but with a higher level of representation. The function of filters specified using this mark-up is easily discernable, but in contrast to other high-level languages such as ANQL [14], our filter mark-up language does not depend upon a pre-defined or standardised naming scheme for packet fields. Figure 2 illustrates a filter that intercepts HTTP requests over IPv4.

LARA++ does not restrict the way in which code is loaded onto active routers. Any component with the authority to instantiate other components may do so. This flexibility allows code to be instantiated in various ways, namely from a local copy of the component already held on the active router, by explicit out-of-band code loading techniques, or using in-band code directly from packets traversing the node. As part of the LARA++ project, we have developed a generic active service discovery and deployment protocol (ASDP) [15], which is an open "pluggable" protocol to support user-definable out-of-band code loading mechanisms.

```
<Filter Name="HTTP-FILTER" InstallationNode="TCP">
    <Rule>
        <Requirement Name="IP-Protocol" Type="Pattern"
          Length="1" BitField="40" BitMask="f0"
          Offset="Focus{IP}"/>
        <Requirement Name="HTTP-Port" Type="Pattern"
          Length="2" BitField="00 50" BitMask="ff ff"
          Offset="This + 2"/>
        <MatchingRule Rule="IP-Protocol && HTTP-Port"/>
    </Rule>
    <Processing Operation=<FilterId> AccessMode="rw"/>
</Filter>
```

Fig. 2. Filter mark-up to intercept HTTP requests carried by an IPv4 transport.

As a result of the LARA++ safety model, which restricts the execution of untrusted code to isolated processing environments, LARA++ does not impose any restrictions in terms of programming languages. As a consequence, active components can be implemented in the developer's favourite programming language (for example, C, C++, C# or Java). All that is needed to support yet another programming language is an implementation of the compact LARA++ programming API as a library acceptable to the target language development tools. This feature also allows component developers to utilise existing language and platform debugging tools (for example, Microsoft Visual Studio.Net) to debug components running live on top of LARA++ as packets flow through them. Due to the often complicated nature of network software, and particularly the semantic leap from conventional to active networks, we believe that this is an invaluable feature.

3 Experimentation with Real-Life Active Network Services

In order to demonstrate the ease with which LARA++ active components can be developed and deployed, we have implemented a number of active components with tasks ranging from remote packet capture (i.e. a "tcpdump" for remote routers) to complex packet routing components. In this section, we detail the development and operation of some of these components.

3.1 Fast Handoff for Mobile-IPv6

Mobile-IPv6 [16] is a technology, recently approved for RFC, which allows network hosts (mobile nodes) to move between subnets whilst remaining addressable via their permanent address (home address). The main functionality is based on the existence of an agent node (home agent) at the mobile node's home network that is responsible for intercepting all traffic destined to the mobile node and relaying it (using tunnels) to the mobile's current network address (care-of address) in the visiting network.

As a routing optimisation, the communication peer (correspondent node) can be informed of the mobile node's care-of address (using a message called a binding

update), which allows it to send packets directly to the mobile's actual network location rather than through the home agent. While this optimization is necessary to prevent scalability issues and improve efficiency, it leads to an awkward situation when the mobile node migrates to a new subnet. Upon configuring its new address, the mobile node will send a binding update to the correspondent node informing it of its new care-of address. Since the correspondent cannot address packets to the new care-of address until the binding update has arrived, packets will still be arriving on the subnet of the old care-of address for at least one round-trip time between the mobile node and the correspondent. These packets would be lost unless the mobile node still had an interface configured on that subnet, which is typically not the case.

Using LARA++ active routers on our mobile access network, we can circumvent this deficiency by augmenting the handoff process using an active component installed at a router close to the handoff. The active component performs "re-routing" of traffic destined to the mobile node's old care-of address to the new care-of address.[2] This simple optimisation reduces the delay between the mobile node sending the binding update and packets being received at the new care-of address to almost nothing, and more importantly greatly reduces the packet loss during handoff procedure. The basic idea of this mechanism along with more details concerning its operation has been previously published by the authors [17].

This Mobile-IPv6 handoff optimisation can be deployed in two ways on an active router: pre-emptively or on-demand. In the pre-emptive scenario, a network administrator or service provider might roll out a component onto strategically chosen active routers as a service to network users. The component would listen for binding updates from all machines on the subnets, and perform the routing optimisation when it detects a mobile node handing off. In the on-demand scenario, the mobile node would dynamically instantiate a component to redirect packets on a convenient active router after it had detected it had moved, or before it moved if it had advance warning (for example, from layer 2 drivers). LARA++ can support both approaches.

Implementing the Mobile-IPv6 handoff optimisation as a LARA++ active component is conceptually quite simple. Essentially, our implementation uses the pre-emptive approach and installs one packet filter to intercept all home address options that pass through the node. A typical LARA++ configuration would not allow just anybody to install such a wide-ranging filter, so that is why this approach is only suitable for a network administrator. Untrusted users should only have permission to install filters on their own flows of packets.

Some implementations of Mobile-IPv6 allow a different care-of address to be used for the same home address when talking to different correspondents. This is useful for multi-homed devices where the routing mechanism decides that the most efficient route to two different correspondents may be via two different interfaces, but it adds additional complexity to the handoff optimisation component. It requires the component to keep track of each correspondent node and mobile node pair. This scalability issue could be a reason why active router administrators may prefer individual users to install the component for their own flows rather than to provide the handoff optimisation service to all users.

Once a home address option is intercepted, the active component can keep a record of the home address binding for that correspondent. If, at some point, it sees a new

[2] This route optimisation is only performed during the transition period, between the mobile node handing-off and the correspondent node adopting the new address for traffic.

care-of address is being used for the correspondent address - home address pair (i.e. a handoff took place), the handoff optimisation component dynamically inserts a short-lived filter for the reverse route which intercepts all packet from the correspondent to the mobile node's old care-of address. For the lifetime of the filter, packets that are intercepted can be re-routed to the correct (new) care-of address thus avoiding packet loss caused by the latency of the round-trip.

```
if (LInsertClassificFilter(ACID,BINDING_UPDATE_PACKET))
   // error handling
while (component_active) {
   if (! LGetPackets(ACID, &Packets, &PacketCount))
        // error handling
   for (Packet=Packets; Packet; Packet=Packet->Next){
        if (Packet->Filter == BINDING_UPDATE_PACKET)
             // Process Home Address Option
        else if (Packet->Filter == PACKET_TO_REDIRECT)
             // Redirect Packet
   }
   if (! LReturnPackets(ACID, Packets))
        // error handling
}
```

Fig. 3. Source code fragment showing the processing loop of the fast handoff component

The source code fragment in figure 3 shows the packet processing loop of the handoff optimisation component. It illustrates the ease with which packets can be intercepted (filtered) and transparently processed by means of LARA++ components.

Figure 4 presents the measurements taken on one of our test LARA++ routers[3] while running the handoff optimisation service. The graph shows the number of packets that would have been lost without the handoff optimisation component (or in other words the number of packets that our component had re-directed and hence saved from being lost) for a range of different media streams (for example, GSM, PCM audio, DivX, MPEG1). The results indicate that the number of packets lost due to the propagation delay of the binding update is proportional to the round trip delay between the active router at the edge of the mobile node's network and its correspondent node. Providing this service in a mobile access network, where the majority of handoffs are localised, can therefore greatly improve handoff performance for mobile devices communicating with distant (in terms of delay) nodes.

This example shows LARA++ is an ideal platform on which to roll out such services as it allows components to operate transparently to the users of the network. In can vary depending on the needs of the network users and the types of traffic flowing through addition, it allows new services to be instantiated dynamically and operate flexibly so the overall service composite the network.

Other ways of addressing the packet loss caused by the end-to-end latency on mobile devices and their correspondents during network handoffs commonly rely upon middle boxes that must be pre-configured to act in a specific way. For example, Hierarchical Mobile-IPv6 [18] uses a "MAP" node to localise handoffs and thus reduce the end-to-end delay of the binding updates, and Fast Handovers for Mobile

[3] A LARA++ active router running on a Windows XP based PC with a 1.7 GHz Pentium 4 processor, 384 MB RAM and three Fast Ethernet Interface Cards.

IPv6 [19] uses extra infrastructure in the network to initiate handovers and establish routing tunnels to the new location of the mobile node. Indeed, had our fast Mobile IPv6 handoff component not been implemented as an active service, it too would have required a dedicated, pre-configured middle box to host the service. A key strength of LARA++ is that it can be used as a basis for the implementation of any of these three mobility solutions and even allows them all to co-exist on the same LARA++ box. Moreover, it enables the dynamic roll-out of all of these services as they are released.

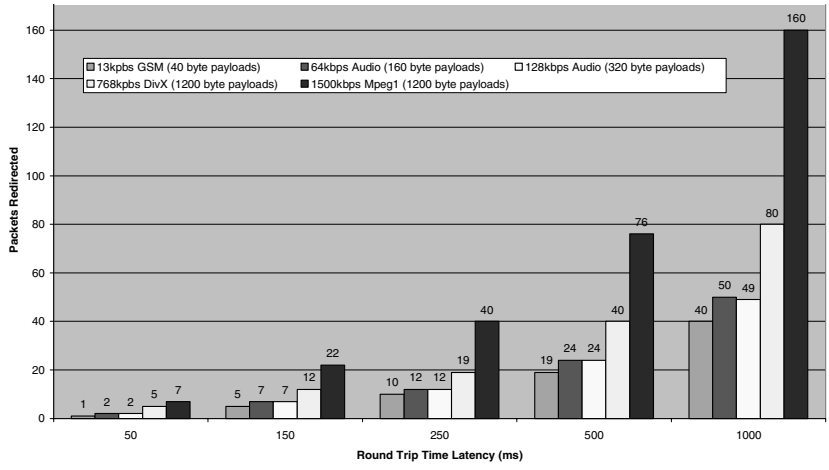

Fig. 4. Measurements of packet loss during Mobile-IPv6 handoff in various network latency and traffic load conditions.

3.2 Network Address Translation

Network Address Translation [20] allows hosts on a non-routable IP network to communicate with other hosts on the Internet via a transparent middle box that translates the private addresses to public addresses for outgoing packets, and vice versa for incoming packets. We implemented NAT as a LARA++ active component in order to further demonstrate the flexibility and range of applicability of LARA++.

The implementation installs packet filters to intercept all inbound IP traffic from the "inside" network of the router (i.e. on each interface on the private network). Upon intercepting every packet using this filter, the active component looks for an existing association between the private address and port, and the correspondent address and port. If no such association for the connection exists, the component creates a new association and assigns it a valid global address and port from a pool. For new connections, the active component then installs another filter for the reverse path from the correspondent address and port to the global address and port for inbound traffic to the "outside" interface. These filters, although numerous, exploit LARA++'s scalable flow filters [11], enabling many thousand of such filters to be installed without adversely affecting the performance. The component is also able to insert a filter so that it can intercept and respond to ARP requests for any addresses from the address pool that the router uses for NAT.

From then on, the NAT component simply intercepts incoming packets on both the inside and outside interfaces (before they reach the standard TCP/IP stack). This allows the active component to co-operate with the standard TCP/IP stack in order to route the packets. Packets intercepted on the inside interface have their source address and port replaced with the global address for the connection, and packets intercepted on the outside interface have their destination address and port swapped for the private address for the connection. After the IP, TCP and UDP checksums have been adjusted to accommodate the change packets are released by the component and then reach the standard TCP/IP stack, which then routes the packets as normal.

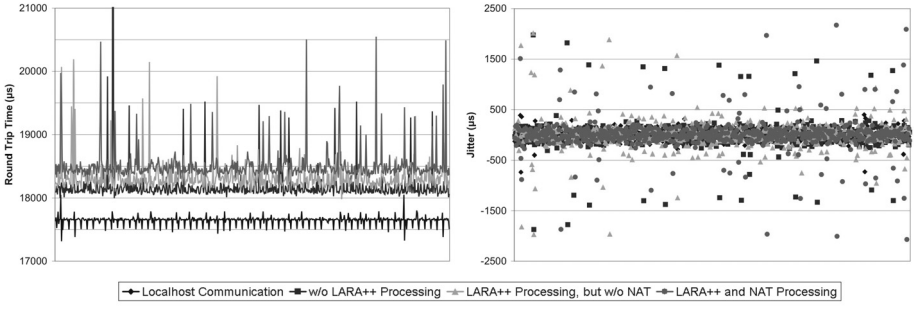

Fig. 5. Performance measurements of packet RTT (left) and jitter (right) of a UDP/IP stream across a LARA++ router with our NAT component instantiated and three other control cases.

We measured the latency and jitter introduced by LARA++ and the NAT component using a UDP-based ping application. The application measured the round-trip delay while sending approximately 60 echo requests per second (see figure 5 – "LARA++ and NAT Processing"). In order to compare the results, we also measured three other configurations: (i) where the LARA++ subsystem was running without the NAT component, (ii) where LARA++ was not running and no NAT was performed, and finally (iii) where UDP communication took place solely on the local host[4].

Of the total 18.5 ms average RTT latency, 17.7 ms is caused by the protocol stacks and the test application at each end of the connection. On average a further 0.4 ms is added by the latency of the actual network. The LARA++ active node OS adds another 0.2 ms to the RTT, and finally the NAT component is responsible for the remaining 0.2 ms. Considering that these latencies can be halved for uni-directional traffic, we can conclude that LARA++ and the NAT component have only a minimal effect on the overall latency.

It is also interesting to compare the jitter on the different experiments. As figure 3 shows, packets to be processed by LARA++ components are more likely to experience additional queuing delays. Consequently, we would expect streams that are processed by LARA++ to have more variance in the packet latencies. However, figure 5 shows that while jitter is increased slightly by LARA++, it remains marginal. For example, while 92.4% of packets have less than 250 µs jitter under normal conditions, with LARA++ and NAT processing this value only drops to 89.7%.

[4] This last control scenario shows the majority of the RTT latency (around 17.7 ms) is caused by the conventional protocol stack and test application, and not by LARA++ or the network.

In order to test the throughput of our test router described in section 3.1, we ran the NAT component on its own with increasing traffic throughput. With 1KByte UDP packets the router was able to sustain approximately 82 Mbps. This compares favourably with a maximum throughput of just over 83 Mbps without LARA++.

3.3 Firewall

Relating back to the previous two examples of active components, while users might legitimately want to install an active component to perform a routing optimization for mobile handoffs on their own flows of packets, there are some applications that could be harmful to the network if end users were able to install them. For example, an application such as NAT installed for a specific user may frustrate the network administrator's attempts to limit access to other networks using conventional means. For example, NAT could be used to bypass security provided by a non-routable network, or some simple variant could be used to bypass port blocking.

This problem is characteristic of a number arising as we deploy active networks. Thus an important lesson highlighted by LARA++ is the need to consider threats to the router by active applications when configuring the router. Just as in conventional routers, configuration of active networks must either systematically prevent undesirable behaviour by active applications or be based upon a well tested "safe" configuration, which is modified only with care.

Users could be prevented from using an active router to circumvent security, as in the case of the flow-specific NAT and port-blocking avoidance examples, by means of a firewall designed to handle such threats. With this in mind, we have implemented a firewall active component designed to be installed and configured in a way that it is tamper-proof to other active components. Much of the difference between this and a conventional firewall is not in the way the component is implemented but rather the way in which it is installed and configured in the classification graph.

It is imperative that unprivileged users have no access to packets, not even their "own" streams, until after a firewall has eliminated packets it regards as undesirable. One way to achieve this is to augment the standard CGT-defined graph with a new classification node that accepts packets immediately prior to the node that processes network layer headers. Also, the classification graph can be configured so ordinary users have access permission only for certain types of packet and no others.

Another consideration is that it is no longer just the network that is the untrusted domain. Filtering out undesirable packets as they are received by the router is no longer a guarantee that all packets forwarded by the router will be desirable. Therefore, a "well-behaved" active router configuration that prevents undesirable behaviour on its networks should also enforce filtering on packets leaving the router.

4 Related Work

The main goal behind LARA++ was to build a highly generic and flexible active router architecture. While we have successfully demonstrated this with a range of very different active services (in section 3), we have not been able to find many related evaluations of active routers in real world deployments. As a consequence, we focus more on a comparison of LARA++ with other active network approaches.

A common objective of most active network approaches is to expedite network evolution through solutions that enable extensibility of network functionality by way of dynamically loaded code. Many active network approaches, such as RouterPlugins [9], Click [8] and the configurable operating system Scout [7] accomplish this through software modules or plug-ins. However, as it has been shown by Hicks and Nettles [21], these approaches all use a fixed "underlying" data structure or program that defines the "glue" for the service composition on the active node. The fact that these composition structures are defined at compile time of the kernel limits extensibility to predefined "slots". LARA++, by comparison, allows the dynamic extension of the classification graph (i.e. allows the creation of new nodes at run-time) and also overcomes the limitation that only one plug-in can be incorporated per slot (i.e. many components can hook packet filters into a classification node).

CANEs/Bowman [22], in contrast, extends the plug-in based approach by a packet filter mechanism that allows the selection of an underlying program, which composes active services provided by the code plug-ins. This flexible classification approach allows Bowman to dynamically deploy new protocols at run-time like LARA++. However, Bowman restricts the selection to a single underlying program; i.e., once an appropriate underlying program has been identified, the service composite is fixed and only dependent on the plug-ins. Furthermore, the literature on CANEs/Bowman implies that only one copy of the packet can be sent to each logical input channel which prevents implicit active program co-operation.

Similarly, VERA [23] appears to be limited in that it allows only one forwarder to be chosen for each packet that is classified. And even if packets could be classified several times, VERA would still lack a method for the chosen forwarders to compose an overall service in a cooperative and deterministic way.

5 Conclusion

The experiences described in this paper have led us to a number of conclusions about LARA++ and about the deployment of active routers in general. LARA++ is a maturing technology suitable for deployment in current IP and IPv6-based networks. It has been designed and built from the ground up with particular focus on interoperability and co-operability with conventional network services on routers, as well as the ease of the development process for network programmers.

We have attempted to demonstrate the flexibility of LARA++ by example. For this reason we developed, deployed and tested three distinct active services through which we have shown that the LARA++ service composition model is sufficiently generic and flexible to provide the different types of services, and allows independent users (role-players) to program the shared network device in a collaborative manner. The measurement results have also indicated that our prototype implementation of LARA++ is suitable in performance for typical edge networks, where value added services and network functionality is often of greater value than only high-speed data forwarding (for example, the fast handoff functionality improves a mobile user's service in a way that clearly outperforms any a high-speed forwarding engine).

However, it is important to note that the range of applications for LARA++ is by no means limited to those mentioned in section 3. The service composition approach based on the classification graph model allows almost any imaginable active network

application, such as media transcoding, protocol experimentation and deployment, routing optimisation, access control, intelligent proxies, and many more.

The most important conclusion we have reached is that LARA++ works! We have successfully deployed LARA++ in real-world situations one would find in today's conventional networks. The service composition model allows multiple applications to co-exists, despite overlapping interests, and contribute towards the overall service composite of the active router. Our implementation of LARA++ has also allowed us to take advantage of existing OS-provided network functionality and interoperate with it. We regard this as essential in allowing a smooth transition from conventional networking towards active networking. This approach leaves open the option of retaining conventional router functionality where the overhead of active solutions cannot be justified by the increase in flexibility.

Our experience developing active components has convinced us that for an active router to be successful, it must offer a very user-friendly interface for developers. This broad requirement encompasses the ability to write active components in many different programming languages, and in particular, the ease of debugging these components. LARA++ does not impose the use of a particular user-space language or development environment, putting active component developers at ease with a familiar development process. LARA++'s filter mark-up language also adheres to this principal with a clean and extensible syntax, making the design of new filters trivial.

Our experiences of deploying LARA++ have taught us a great deal about the trade-off between performance and flexibility. The line speeds we have been able to sustain, whilst good, are far short of those appropriate for core networks, but are acceptable within typical edge networks. Also, we have come to the conclusion that the majority of active network applications are ideally provided at the edge of the network, as this is where the services are required to have the desired effect. For example, the fast Mobile IPv6 handoff component relies on being deployed close to the point of path aggregation between the old and new care-of addresses. So long as handoffs are between topologically close networks, this is almost always at an edge router. Similarly, the NAT component must be installed at the edge of a private network.

Indeed, most active applications are best suited to deployment in edge networks, and in edge networks the flexibility to deploy such functionality is more important than optimal performance. Thus, we see edge routers as the most suitable candidates for augmentation with LARA++ functionality.

Finally, we have also noted that as new network capabilities are deployed, new threats to the network arise. Conventional solutions such as port blocking and firewalling may not completely close the avenues of attack. A secure system of permissions and trust is helpful, but is no substitute for careful and systematic configuration with regard to the new potential threats.

References

1. S. Schmid, J. Finney, A.C. Scott, W.D. Shepherd, "Component-based Active Network Architecture", IEEE Symposium on Computers and Communications, July 2001.
2. B. Schwartz et al., "Smart Packets: Applying Active Networks to Network Management". ACM Transactions on Computer Systems, volume 18(1), pp 67-88, 2000.
3. G. Goldszmidt and Y. Yemini, "Delegated Agents for Network Management". IEEE Communications, volume 36(3), pp 66-70, March 1998.

4. B. Braden et al., “Introduction to the ASP Execution Environment 1.5”. Technical report, http://www.isi.edu/active-signal/ARP/DOCUMENTS/ASP_EE.ps, November 2001.
5. T. Harbaum, A. Speer, R. Wittmann and M. Zitterbart, “Providing Heterogeneous Multicast Services with AMNET”. Communications and Networks, 3(1), March 2001.
6. R. Keller et al., “An Active Router Architecture for Multicast Video Distribution”. In Proc. of IEEE INFOCOM (3), pp 1137-1146, 2000.
7. A. Montz et Al., “Scout: A Communications-Oriented Operating System”, In Operating Systems Design and Implementation, page 200, 1994.
8. R. Morris, E. Kohler, J. Jannotti, M Kaashoek, “The Click Modular Router“, In Proc. of ACM Symposium on Operating Systems Principles, pages 217-231, December 1999.
9. D. Decasper, Z. Dittia, G. Parulkar, B. Plattner, “Router Plug-ins: A Software Architecture for Next Generation Routers”, In Proc. of SIGCOMM, pp 229-240, September 1998.
10. The Caml Language. Online Reference, INRIA, http://caml.inria.fr/.
11. S. Schmid, T. Chart, M. Sifalakis, A. C. Scott, “Flexible, Dynamic and Scalable Service Composition for Active Routers”, In Proc. of IWAN 2002, pp 253-266, December 2002.
12. D.S. Alexander et al., “Active Network Encapsulation Protocol (ANEP)”, July 1997.
13. S. McCann and V. Jacobson, “The BSD Packet Filter: A New Architecture for User-level Packet Capture”. In Proc. of USENIX Conference, Berkeley, 1993.
14. C. M. Rogers, “ANQL – An Active Networks Query Language”. In Proc. of IWAN 2002, Zurich, pp 99-110, December 2002.
15. M. Sifalakis, S. Schmid, T. Chart, D. Hutchison, “A Generic Active Service Deployment Protocol”. In Proc. of ANTA 2003, pp 100-111, Osaka, May 2003.
16. D. Johnson, C. Perkins, J. Arkko, “Mobility Support in IPv6”, Internet Draft draft-ietf-mobileip-ipv6-24.txt, June 2003. WORK IN PROGRESS.
17. S. Schmid, J. Finney, A. C. Scott, W. D. Shepherd, “Active Component Driven Network Handoff for Mobile Multimedia Systems”. In Proc. of IDMS 2000 (pp 266-278), University of Twente, Enschede, The Netherlands, October 2000.
18. H. Soliman, C. Castelluccia, K. El-Malki, L. Bellier, “Hierarchical Mobile IPv6 Mobility Management”, Internet Draft draft-ietf-mobileip-hmipv6-08.txt, June 2003.
19. R. Koodli, “Fast Handovers for Mobile IPv6”, Internet Draft draft-ietf-mobileip-fast-mipv6-07.txt, September 2003. WORK IN PROGRESS.
20. K. Egevang et al., “The IP Network Address Translator (NAT)”, RFC 1631, May 1994.
21. M.W. Hicks and S. Nettles, “Active Networking Means Evolution (or Enhanced Extensibility Required)”, In Proc. of IWAN 2000, October 2000.
22. S. Merugu et al., “Bowman and CANEs: Implementation of an Active Network", In Proc. of 37th Conference on Communication, Control and Computing, September 1999.
23. S. Karlin, L. Peterson, “VERA: An Extensible Router Architecture”. In Journal “Computer Networks (Amsterdam, Netherlands, 1999)” 38(3), pp 277-293, 2002.