An Architecture for Application Layer Routing

Atanu Ghosh

Michael Fry

atanu,mike@socs.uts.uts.au

Faculty of Information Technology University of Technology Sydney,

Building 4, Thomas Street, PO Box 123, NSW 2007 Australia

Jon Crowcroft

jon@cs.ucl.ac.uk

Department of Computer Science,

University College London, Gower Street,

LONDON WC1E 6BT

Keywords: Active Networks, Application Layer Routing

May 2, 2000

Abstract

We have previously proposed, implemented and demonstrated an Application Layer Active Network (ALAN) infrastructure. This infrastructure permits the dynamic deployment of active services in the network, but at the application level rather than the router level. Thus the advantages of active networking are realised, without the disadvantages of router level implementation. However we have previously left unsolved the issue of appropriate placement of ALAN supported services. This is an Application Layer Routing problem. In this paper

we define this problem and show that, in contrast to IP, it is a multi-metric problem. We then propose an architecture that helps conceptualise the problem and build solutions. We propose detailed approaches to the active node discovery and state maintenance aspects of Application Layer Routing (ALR).

1 Introduction

We have previously proposed, implemented and demonstrated an approach to active networks based on Application Layer Active Networking (ALAN) [2]. We believe that this approach can achieve many of the benefits ascribed to active networks without the considerable drawbacks evident in the implementation of active networks in IP routers.

Our approach has been validated by developments in the commercial Internet environment. It is the case that some Internet Service Providers (ISPs) will support servers at their sites supplied by third parties, to run code of their (3rd parties') choice. Examples include repair heads [25], which are servers in the network which help with reliable multicast in a number of ways. In the reliable multicast scenario an entity in the network can perform ACK aggregation and retransmission. Another example is Fast Forward Networks Broadcast Overlay Architecture [26]. In this scenario there are media bridges in the network. These are used in combination with RealAudio [27] or other multimedia streams to provide an application layer multicast overlay network.

We believe that rather than placing "boxes" in the network to perform specific tasks, we should place generic boxes in the network that enable the dynamic execution of application level services. We have proposed a ALAN environment based on Dynamic Proxy Servers (DPS). In our latest release of this system we rename the application layer active nodes of the network Execution Environments for Proxylets (EEPs). By deploying active elements known as *proxylets* on EEPs we have been able to enhance the performance of network applications.

In our initial work we have statically configured EEPs. This has not addressed the issue of appropriate location of application layer services. This is essentially an Application Layer Routing (ALR) problem.

For large scale deployment of EEPs it will be necessary to have EEPs dynamically join a mesh of EEPs, with little to no configuration. As well as EEPs dynamically discovering each other applications that want to discover and use EEPs should also be able to choose appropriate EEPs as a function of one or more form of routing metric. Thus the ALR problem resolves to an issue of overlaid, multi-metric routing.

This paper is organised as follows. We first describe our ALAN infrastructure by way of some application examples. These examples reveal the Application Layer Routing issues that require solution. We then propose an architecture that aids conceptualisation and provides a framework for an implementable solution. This paper concentrates on the issues of node (EEP) discovery and state maintenance. We conclude by referencing related work and describing our future work, which is already in progress.

2 Application Layer Active Networking

Our infrastructure is quite simple and is composed of two components. Firstly we have a proxylet. A proxylet is analogous to an applet [28] or a servlet [29]. An applet runs in a WWW browser and a servlet runs on a WWW server. In our model a proxylet is a piece of code which runs in the network. The second component of our system is an (Execution Environment for Proxylets) EEP. The code for a proxylet resides on a WWW server. In order to run a proxylet a reference is passed to an EEP in the form of a URL and the EEP downloads the code and runs it. The process is slightly more involved but this is sufficient details for the current explanation.

In our initial prototype system, "funnelWeb", the EEP and the proxylets are written in Java. Writing in Java has given us both code portability as well as the security [30] of the sandbox model. The "funnelWeb" [31] package runs on Linux, Solaris and Windows NT.

2.1 WWW cache proxylet

We have written a number of proxylets to test out our ideas. Possibly the most complicated example has been a webcache proxylet [32]. In this example a webcache proxylet is co-resident

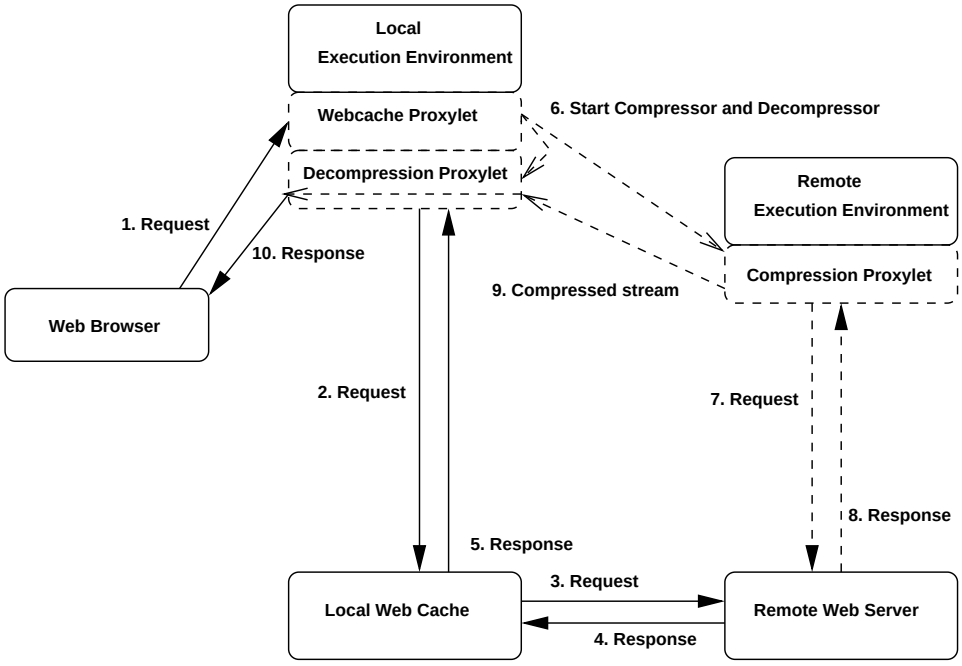

Figure 1: Text Compression

with a squid cache [33]. WWW browsers typically support a configuration option which allows all accesses to go via a cache. A WWW browser could be pointed at our cache. The webcache proxylet does not actually perform any caching, relying instead on the co-resident cache. What it can do however is to perform some transformations on the retrieved page such as transcoding or compression. This can be performed as a function of the mime content type of a requested URL.

For example for large text files it might compress the file before it is transmitted. Rather than do this at the WWW server, we try to locate an EEP which is close to the WWW server. A proxylet is then sent to the EEP, which downloads the page from the WWW server and compresses it. The compressed text is received by a decompressor proxylet launched by the webcache proxylet close to the WWW client. This proxylet decompresses the page and returns it to the WWW browser. In our experiments we were able to compress the data for the transcontinental (low bandwidth) parts of a connection. This process is illustrated in Figure 1.

A limitation of our initial experiments is that the locations of the various EEPs is known a priori by the webcache proxylet. Another problem was that it is not always clear that it is useful

to perform any level of compression. For example if a WWW server is on the same network as the browser it may make no sense to attempt to compress transactions.

¿From this application of proxylets two clear requirements emerge which need to be satisfied by our application layer routing infrastructure. The first requirement is to return information regarding proximity. A question asked of the routing infrastructure may be of the form: return the location of an EEP which is close to a given IP address. Another requirement could be the available bandwidth between EEPs, as well as perhaps the bandwidth between an EEP and a given IP address. An implied requirement also emerges. It should not take more network resources or time to perform ALR than to simply perform the native transaction.

Given this location and bandwidth information the webcache proxylet could now decide if there were any benefit to be derived by transcoding or compressing a transaction. So a question asked of the application layer routing infrastructure may be of the form: find a EEP "close" to this network and also return the available bandwidth between that EEP and here.

2.2 TCP bridge

One of our simplest proxylets is a TCP bridge proxylet. The TCP proxylet runs on an EEP and accepts connections on a port that is specified when the proxylet is started. As soon as a connection is accepted a connection is made to another host and port. This proxylet allows application layer routing of TCP streams. It could obviously be extended to route specific UDP streams.

We have experienced the benefits of a TCP bridge by using telnet to remote log in to computers across the globe. A direct telnet from one computer to another across the global Internet may often experience very poor response times. However if one can chain a number of TCP connections (essentially, source routing) by logging into intermediate sites, better performance is typically achieved. This is because the segmented TCP connections respond more promptly to loss, and do not necessarily incur the overhead of end-to-end error control.

A requirement that emerges from this scenario is the need for the application routing infras-

tructure to return a number of EEPs on a particular path. We may ask of the ALR: give me a path between node A and node B on the network which minimises delay. We may also ask for a path between node A and B which maximises throughput, by being more responsive to errors. It may also be possible that we require more than one path between node A and node B for fault tolerance.

2.3 VOIP gateway

A proxylet that we intend to write is co-located with a gateway from the Internet to the PSTN. The idea is that a person is using their PDA (Personal Digital Assistant) with a wireless network interface such as IEEE 802.11 or perhaps Bluetooth [34].

Using a packet audio application such as "vat", the user wishes to make a voice call to a telephone via an IP-to-telephony gateway. The simple thing to do would be to discover a local gateway. More interesting might be to find the closest gateway to the telephony endpoint. An argument for doing this might be that it is cheaper to perform the long haul part of the connection over the Internet rather than by using the PSTN.

This adds another requirement: the ability to discover information about available services. The normal model for service discovery is to find a service in the local domain. We have a requirement for service discovery across the whole domain in which EEPs are running. So a proxylet on an EEP which is providing a VOIP gateway may want to inject information into the routing infrastructure which can be used by VOIP aware applications.

2.4 Multicast

With the seeming failure of wide area multicast deployment it starts to make sense to use proxylets inside the network to perform fanout of streams, as well as perhaps transcoding and retransmission. A requirement that emerges for multicast is that there is enough information in the routing infrastructure for the optimal placement of fanout points.

3 Application Layer Routing Architecture

A number of routing requirements for our ALAN infrastructure have emerged from the examples described in the previous section. In essence our broad goal is to allow clients to choose an EEP or set of EEPs on which to run proxylets based on one or more cost functions. Typical metrics will include:

- Available network bandwidth.
- Current delay.
- EEP resources.
- Topological proximity.

We may also want to add other constraints such as user preferences, policy, pricing, etc. In this paper we focus initially on metric-based routing.

An Application Layer Routing (ALR) solution must scale to a large, global EEP routing mesh. It must permit EEPs to discover other EEPs, and to maintain a notion of "distance" between each EEP in a dynamic and scalable manner. It will allow clients to launch proxylets (or "services") based on one or more metric specifications and possibly other resource contingencies. Once these services are launched, they become the entities that perform the actual "routing" of information streams.

We therefore propose a ALR architecture that has four components. In this section we simply provide an overview of the architecture. The four components are as follows.

1. EEP Discovery.
2. Routing Exchanges.
3. Service Creation.
4. Information Routing.

EEP discovery is the process whereby an EEP discovers (or is able to discover) the existence of all other EEPs in the global mesh. In our current implementation (described below) all EEPs register at a single point in the network. This solution is clearly not scalable, requiring the introduction of a notion of hierarchy. Our proposed approach is described in the next section. The approach addresses both the arrival of new EEPs and the termination or failure of existing EEPs.

Routing exchanges are the processes whereby EEPs learn the current state of the ALAN infrastructure with regard to the various metrics. On this basis EEP Routing Tables are built and maintained. The routing meshes embedded in routing tables may also implement notions of clusters and hierarchy. However these structures will be dynamic, depending on the state of the network, with different structures for different metrics. The state information exchanges from which routing information is derived may be explicitly transmitted between EEPs, or may be inferred from the observation of information streams.

Service creation is the process whereby a proxylet or set of proxylets are deployed and executed on one or more EEP. The client of this service creation service specifies the proxylets to be launched and the choice(s) of EEP specified via metrics. The client may also specify certain service dependencies such as EEP resource requirements.

The service proxylets to be launched depend entirely on the service being provided. They encompass all the examples described in the previous section. For example, the webcache proxylet launches transcoders or compressors according to mime content type. The metric used here will be some proximity constraint (e.g. delay) to the data source, and/or available bandwidth on the path. The TCP bridge proxylets will be launched to optimise responsiveness to loss and maximise throughput. The VOIP gateway proxylet will require a telephony gateway resource at the EEP.

Information routing is the task performed by the proxylets once launched. Again the function performed by these proxylets are dependent on the service. It may entail information transcoding, compression, TCP bridging or multicast splitting. In each case information is forwarded to the next point(s) in an application level path.

The rest of this paper is devoted to describing our more detailed proposals for EEP Discovery and Routing Exchanges.

4 Discovery

4.1 Discovery phase

We will now describe how the discovery phase takes place. The discovery phase is implemented by a "discovery proxylet" that is pre-configured with each EEP. Since proxylets are loaded on EEPs by URL reference, it is trivial to update the version of the discovery proxylet - it will be automatically loaded when a EEP starts up.

The function of the discovery phase is for all EEPs to join a global "database", which can be used/interrogated by a "routing proxylet" (discussed further in the next section) to find the location of an EEP(s) which satisfies the appropriate constraints. Constructing this database through the discovery phase is the first stage towards building a global routing infrastructure.

4.1.1 Requirements

There a number of requirements for our discovery phase:

1. The solution should be self configuring. There should be no special configuration such as tunnels between EEPs.
2. The solution should be fault tolerant.
3. The solution should scale to hundreds or perhaps thousands of deployed EEPs.
4. No reliance on technologies such as Multicast.
5. The solution should be flexible.

4.1.2 The discovery protocol

The solution involves building a large distributed database of all nodes. A naive registration/discovery model might have all registrations always going to one known location. However it is obvious that such a solution would not scale beyond a handfull of nodes. It would not be suitable for a global mesh of hundreds or thousands of nodes.

In order to spread the load we have opted for a model where there is a hierarchy. Initially registrations may go to the root EEP. But a new list of EEPs to register with is returned by the EEP. So a hierarchy is built up. An EEP has knowledge of any EEPs which have registered with it as well as a pointer to the EEP above it in the hierarchy. So the information regarding all the EEPs is distributed as well as distributing where the registration messages go. The time to send the next registration message is also included in the protocol. So perhaps as the number of EEPs grows or as the system stabilises the frequency of the messages can be decreased.

If an EEP that is being registered with fails then the EEP registering with it will just try the next EEP in its list until it gets back to the root EEP.

With this hierarchal model, if the list of all EEPs is required then an application (normally this will be only routing proxylets), can contact any EEP and send it a node request message. In response to a node request message number three chunks of information will be returned. A pointer up the hierarchy where this EEP last registered. A list of EEPs that have registered with this EEP if any. A list of the backup EEPs that this EEP might register with. Using the node message interface it is possible to walk the whole hierarchy. So either an application extracts the whole table and starts routing proxylets on all nodes. Or a routing proxylet is injected into the hierarchy which replicates itself using the node message.

The discovery proxylet offers one more service. It is possible to register with the discovery proxylet to discover state changes. The state changes that are of interest are a change in where registration messages are going, a EEP which has failed to re-register being timed out, and a new EEP joining the hierarchy.

In the discussion above we have not said anything about how the hierarchy is actually con-

structed. We don't believe that it actually matters as long as we have distributed the load, to give us load balancing and fault tolerance. It may seem intuitive that the hierarchy be constructed around some metric such as RTT. So say all EEPs in the UK register with an EEP in the UK. This would certainly reduce the number of packets exchanged against a model that had all the registrations made by EEPs in the UK going to Australia. A defence against pathological hierarchies is that the registration time can be measured in minutes or hours not seconds. We can afford such long registration times because we expect the routing proxylets to exchange messages at a much higher frequency and form hierarchies based on RTT and bandwidth etc... So a failed node will be discovered rapidly by the routing level. Although it seems like a chicken and egg situation the discovery proxylets could request topology information from the routing proxylets. To aid in the selection of where a new EEP should register. We also don't want to repeat the message exchanges that will go on in the higher level routing exchanges.

We have considered two other mechanisms for forming hierarchies. The first is a random method. In this scheme a node will only allow a small fixed number of nodes to register with it. Once this number is exceeded any registration attempts will be passed to one of the list of nodes which is currently registered with the node. The selection can be made randomly. If for example the limit is configured to be five as soon as the sixth registration request comes in the response will select one of the already registered nodes as the new parent EEP to register with. This solution will obviously form odd hierarchies in the sense that they do not map onto the topology of the network. It could however be argued that this method may give an added level of fault tolerance.

The second method that we have been considering is a hierarchy based on domain names, so the hierarchy maps directly onto the DNS hierarchy. Thus all sites with the domain ".edu.au" all register with the same node. In this case we can use proximity information derived from DNS to build the hierarchy. This scheme will however fail with the domain ".com". Where nothing can be implied about location. The node that is accepting registrations for the ".com" domain will be overwhelmed.

We believe that, since registration exchanges occur infrequently, we can choose a hierarchy

forming mechanism which is independent of the underlying topology. The more frequent routing exchanges discussed in the next section will map onto the topology of the network and catch any node failures.

We have discussed a hierarchy with a single root. If this proved to be a problem, we could have an infrastructure with multiple roots. But unlike the rest of the hierarchy the root nodes would have to be aware of each other through static configuration, since they would have to pool information in order to behave like a single root.

4.1.3 Messages exchanged by discovery proxylet

The types of messages used are registration messages and node messages. The registration messages are used solely to build the hierarchy. The node messages are used to interrogate the discovery infrastructure.

Version numbers are used to both detect a protocol mismatch, and to trigger the reloading of a new version of the discovery proxylet. The host count will make it simple to discover the total number of nodes by just interrogating the root node.

- Registration request message.
  - Version number.
  - This nodes name.
  - Count of hosts registered below this node.
- Registration acknowledgement message,

  sent in response to a registration request message.
  - Version number.
  - Next registration time.

    A delay time before the next registration message should be sent. This timer can be adjusted dynamically as a function of load, or reliability of a node. If a node has many

children the timer may have to be increased to allow this node to service a large number of requests. If a child node has never failed to register in the required time, then it may be safe to increase the timeout value.

  - List of nodes to register with,

    used for fault tolerance. Typically the last entry in the list will be the root node.

- Node request message.

  - Version number.

- Node acknowledgement message, sent in response to a node request message.

  - Version number.
  - Host this node registers with.
  - List of backup nodes to register with.

    It is useful to have the list of backup nodes in case the pointer up to the local node fails, while a tree walk is taking place.

  - List of nodes that register with this node.

5 Routing exchanges

Once the underlying registration infrastructure is in place this can be used to start routing proxylets on the nodes. The process is simple and elegant. A routing proxylet can be loaded on any EEP. One routing proxylet needs to be started on just one EEP anywhere in the hierarchy. By interrogating the discovery proxylet all the children can be found as well as the pointers up the tree. The routing proxylet then just starts an instance of itself on every child and on its parent. This process repeats and a routing proxylet will be running on every EEP. The routing proxylet will also register with the discovery proxylet to be informed of changes (new nodes, nodes disappearing, change in parent node). So once a routing proxylet has launched itself across the network, it can track changes in the network.

Many different routing proxylets can be written to solve various problems. It may not even be necessary run a routing proxylet on each node. It may be possible to build up a model of the connectivity of the EEP mesh by only occasionally having short lived, probing proxylets running on each cluster.

The boundary between having a centralised routing infrastructure against a distributed routing infrastructure can be shifted as appropriate.

We believe that many different routing proxylets will be running using different metrics for forming topologies. Obvious examples would be paths optimised for low latency. Or paths optimised for high bandwidth. This is discussed further below.

5.1 Connectivity mesh

In a previous section we described a number of proxylets that we have already built and are considering building, along with their routing requirements. Some routing decision can be solved satisfactorily by using simple heuristics such as domain name. There will however be a set of services which require more accurate feedback from the routing system.

We believe that some of the more complex routing proxylets will have to make routing exchanges along the lines of map distribution MD [7]. A MD algorithm floods information about local connectivity to the whole network. With this information topology maps can be constructed. Routing computations can be made hop by hop, an example map be an algorithm to compute the highest bandwidth pipe between two points in the network. A more centralised approach may be required for perhaps application layer multicast where optimal fan out points are required.

In fixed routing architectures each node, by some mechanism, propagates some information about itself and physically connected neighbours. Metrics such as bandwidth or RTT for these links will also be included in these exchanges. In the ALR world we are not constrained by using only physical links to denote neighbours. There may be conditions where nodes on opposite sides of the world may be considered neighbours. Also links do not necessarily need to be bidirectional. We expect to use various metrics for selecting routing neighbours. We won't necessarily be distributing

multiple metrics through one routing mesh. We may create a separate routing mesh for each metric. This is explored further below.

Another important issue which doesn't usually arise from traditional routing is that if care is not taken, certain nodes may disappear from the routing cloud if a node cannot find a neighbour.

5.2 Snooping for network state

The maintenance of network state can be performed via periodic exchanges between routing proxylets. While this has merit, it has the disadvantage that the exchanges themselves put load on the network, and therefore impact network state. While not dismissing this approach, we propose here an alternative approach that uses more implicit routing exchanges.

Proxylets started on EEPs make use of standard networking API's to transmit information. It would be relatively simple to add a little shim layer, such that whenever a networking call is made we can estimate, for example, the bandwidth of a path. Thus bandwidth information derived from service proxylets such as a multicast proxylet can be feed back into the routing infrastructure. An initial proposal for how this might be utilised is now discussed.

5.3 Self Organising Application-level Routing - SOAR

We propose a recursive approach to this, based on extending ideas from RLC[13] and SOT[12], called Self Organised Application-level Routing (SOAR). The idea is that a soar does three tasks:

1. Exchanges graphs/maps [7][8] with other SOARs in a region use traffic measurement to infer costs for edges in graphs re-define regions.

   A region (and there are multiple sets of regions, one per metric), is informally defined as a set of SOARs with comparable edge costs between them, and "significantly" different edge costs out of the region. An election procedure is run within a region to determine which SOAR reports the region graph to neighbour regions. Clearly, the bootstrap region is a single SOAR.

W_m ; Maximum advertised receive window
RTT ; The Round Trip Time
b ; the number of packets acknowledged by 1 ACK
p ; the mean packet loss probability
B ; the throughput achieved by a TCP flow

Table 1: Terms in the Padhye TCP Equation

The definition of "significant" above needs exploring. An initial idea is to use the same approach as RLC[13] - RLC uses a set of data rates distributed exponentially - typically, say, `10kbps, 56kbps, 256Kbps, 1.5Mbps, 45Mbps` and so on.

This roughly corresponds to the link rates seen at the edge of the net, and thus to a set of users possible shares of the net at th next "level up". Fine tuning may be possible later.

2. SOAR uses measurement of user traffic (as in SOT[12]) to determine the available capacity - either RTCP reports of explicit rate, or inferring the available rate by modelling a link and other traffic with the Padhye[5] equation provide ways to extract this easily. Similarly, RTTs can be estimated from measurement or reports (or an NTP proxylet could be constructed fairly easily). This idea is an extension of the notion proposed by Villamazar[9] using the Mathis[4] simplification, in "OSPF Optimized Multipath", $p < (MSS/(BW * RTT))^2$

   A more complex version of this was derived by Padhye et al.:

$$B = min(\frac{W_m}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)}) \quad (1)$$

   RTT Is estimated in the usual way, if there is two way traffic. $R\bar{T}T_i = RTT_i * alpha + (1 - alpha) * R\bar{T}T_{i-1}$ It can also be derived using NTP exchanges.

   Then we simply measure loss probability (p) with a EWMA. Smoothing parameters (alpha, beta for RTT, and loss averaging period for p) need to be researched accurately - note that a lot of applications use this equation directly now[6] rather than AIMD sending a la TCP.

This means that the available capacity (after you subtract fixed rate applications like VOIP) is well modelled by this.

3. Once a SOAR has established a metric to its bootstrap configured neighbour, it can declare whether that neighbour is in its region, or in a different region - as this continues, clusters will form. The neighbour will report its set of "neighbour" SOARs (as in a distance vector algorithm) together with their metrics (strictly, we don't need the metrics if we are assuming all the SOARs in a region are similar, but there are lots of administrative reasons why we may - in any case, a capacity-based region will not necessarily be congruent with a delay-based region. Also, it may be useful to use the neighbour exchanges as part of the RTT measurement).

   The exchanges are of region graphs or maps - each SOAR on its own forms a region and its report is basically like a link state report. We should explore whether the SOAR reports should be flooded within a region, or accumulated as with a distance vector.

   A graph is a flattened list of node addresses/labels, with a list of edges for each node, each with one or more metrics.

   Node and Edge Labels are in a URL-like syntax, for example `soar://node-id.region-id.soar-id.net` and an Edge label is just the far end node label.

   Metrics are `<type, value>` tuples (ASCII syntax). Examples of metrics include:

   - A metric for delay is typically milliseconds
   - A metric for throughput is Kbps
   - A metric for topological distance is hop count
   - A metric for topological neighbour is IP address/mask

   As stated above, a SOAR will from time to time discover that a neighbour SOAR is in a different region. At this point, it marks itself as the "edge" of a region for that metric. This is an opportunity for scaling - the SOARs in a region use an election procedure to determine

which of them will act on behalf of the region. The elected SOAR (chosen by lowest IP address, or perhaps at the steiner centre, or maybe by configuration), then pre-fixes the labels with a region id (perhaps made up from date/time and elected SOAR node label.

```
soar://node-id.region-id.soar-id.net/metricname
soar://node-id.region-id.region-id.soar-id.net/metricname
soar://node-id.region-id.region-id.region-id.soar-id.net/metricname
etc
```

6 Current Status

We have a prototype system "funnelWeb 2.0.1" [31], which supports the deployment of proxylets. We are running EEPs at a number of Universities in the UK as well as at UTS. We have a very basic combined discovery and routing proxylet which supports the notion of proximity. A DNS name can be given to the routing proxylet and a EEP close to the name will be returned. The first cut at this problem just matches on DNS names. This very crude approximation works surprisingly well, for example it is able to find a EEP close to a web server in our text compression example.

We have a test page at <URL:http://dmir.socs.uts.edu.au/projects/alpine/routing/index.html> . One of the advantageous features of writing the discovery and routing entities as proxylets has been that we can totally redeploy the whole infrastructure in a very short time.

We are in the process of rewriting our web cache proxylet [32], to make use of the routing infrastructure. An implementation of the RTT and bandwidth estimators for SOAR is underway.

7 Related Work

Active Networks has been an important area of research since the seminal paper by Tennenhouse et al.[1] This paper is based on work in a research project which more oriented towards active

services[2][3], which has emerged as an important subtopic though the Openarch conference and related events.

In the current work, we are attempting to address problems associated with self-organisation and routing, both internal to Active Services infrastructure, as well as in support of specific user services. To this end we are taking a similar approach to the work in scout[22], specifically the joust system[23], rather than the more adventurous, if less deterministic approach evidenced in the Ants work[19][20]. Extension of these ideas into infrastructure services has been carried out in the MINC Project[10] and is part of the RTP framework (e.g. Perkins work on RTP quality[11].

Much of the work to date in topology discovery and routing in active service systems has been preliminary. We have tried to draw on some of the more recent results from the traditional (non active) approaches to these two sub-problems, and to this end, we have taken a close look at work by Francis[18], as well as the nimrod project[7].

Our approach is trying to yield self-organising behaviour as in earlier work[12][20][24], as we believe that this is attractive to the network operator as well as to the user.

There have been a number of recent advances in the area of estimation of current network performance metrics to support end system adaption, as well as (possibly multi-path) route selection, e.g. for throughput, there is the work by Mathis[4], Padhye[5] and Handley[6], and for multicast[13], and its application in routing by Villamizar[9]. more recently, several topology discovery projects have refined their work in estimating delays, and this was reported in Infocom this year, for example, in Theilman[14], Stemm[15], Ozdemir[16] and Duffield[17].

8 Future Work

The next stage of the research is to experiment by implementation with different discovery and routing exchange mechanisms. Implementing these as proxylets makes experimental deployment relatively simple. We already have a small (in node numbers) global network of EEPs which permits live experimentation. We are hoping that more EEPs can be located in different countries

and continents in the near future (the authors are happy to talk to any groups that may be willing to host EEPs).

As mentioned above, we already have implementations of bandwidth and RTT extimators for SOAR underway. These are building on previous work that has implemented a reliable muticast protocol in the form of proxylets [13]. Likewise we intend to build service proxylets based on our Application Layer Routing infrastructure. These are likely to implement multicast for media streaming, and link to related work that we have underway that addresses caching within this infrastructure [32].

References

[1] "Towards an Active Network Architecture", D. L. Tennenhouse and D. J. Wetherall, ACM Computer Communication Review, vol. 26, no. 2, pp. 5–18, Apr. 1996.

[2] Application Level Active Networks Michael Fry and Atanu Ghosh, UTS, in Computer Networks and ISDN Systems, `http://dmir.socs.uts.edu.au/projects/alan/papers/cnis.ps`

[3] Elan Amir, Steven McCanne and Randy Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding," ACM Computer Communication Review, vol. 28, no. 4, pp. 178–189, Sep. 1998

[4] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. "The macroscopic behaviour of the TCP congestion avoidance algorithm." ACM Computer Communication Review, 27(3), July 1997.

[5] Jitendra Padhye, Victor Firoiu, Don Towsley and Jim Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," ACM Computer Communication Review, vol. 28, no. 4, pp. 303–314, Sep. 1998.

[6] An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet R. Rejaie, M. Handley, D. Estrin. Proc. Infocom 99 `http://www.aciri.org/mjh/rap.ps.gz`

[7] New Internet Routing (a.k.a nimrod) `http://ana-3.lcs.mit.edu/ jnc/nimrod/docs.html`

[8] Paul Francis PhD thesis, UCL 1992. `ftp://cs.ucl.ac.uk/darpa/pfrancis-thesis.ps.gz`

[9] Curtiz Villamizar, Work in progress, i-d - ftp from http://www.ietf.org/ietf `draft-ietf-ospf-omp-02.txt`

[10] Multicast Inference of Network Congestion (minc) `http://gaia.cs.umass.edu/minc`

[11] RTP Quality Matrix `http://www-mice.cs.ucl.ac.uk/multimedia/software/rqm/`

[12] Self Organising Transcoders (sot) Isidor Kouvelas et al NOSSDAV 1998 f`tp://cs.ucl.ac.uk/darpa/sot.ps.gz`

[13] Receiver Driven Layered Congestion Control (a.ka. rlc) `ftp://cs.ucl.ac.uk/darpa/infocom98.ps.gz` and `ftp://cs.ucl.ac.uk//darpa/rlc-dps.ps.gz`

[14] Dynamic Distance Maps of the Internet Wolfgang Theilmann (University of Stuttgart), Kurt Rothermel (University of Stuttgart) Proceedings of IEEE Infocom 2000.

[15] A Network Measurement Architecture for Adaptive Applications Mark Stemm (University of California at Berkeley), Srinivasan Seshan (IBM T.J. Watson Research Center), Randy H. Katz (University of California at Berkeley) Proceedings of IEEE Infocom 2000.

[16] Scalable, Low-Overhead Network Delay Estimation Volkan Ozdemir (North Carolina State University), S. Muthukrishnan (ATT Labs - Research), Injong Rhee (North Carolina State University)

[17] Multicast Inference of Packet Delay Variance at Interior Network Links Nick Duffield (ATT Labs - Research), Francesco Lo Presti (ATT Labs - Research and University of Massachusetts)

[18] YALLCAST arhictecture, Paul Francis http://www.aciri.org/yoid/docs/index.html

[19] David Wetherall, Ulana Legedza and John Guttag, "Introducing New Internet Services: Why and How," IEEE Network, vol. 12, no. 3, pp. 12–19, May 1998.

[20] Maria Calderon, Marifeli Sedano, Arturo Azcorra and Cristian Alonso, "Active Network Support for Multicast Applications," IEEE Network, vol. 12, no. 3, pp. 46–52, May 1998.

[21] D. Scott Alexander, William A. Arbaugh, Michael W. Hicks, Pankaj Kakkar, Angelos D. Keromytis, Jonathan T. Moore, Carl A. Gunter, Scott M. Nettles and Jonathan M. Smith, "The SwitchWare Active Network Architecture," IEEE Network, vol. 12, no. 3, pp. 27–36, May 1998.

[22] A. Montz, D. Mosberger, S. O'Mealley, L. Peterson, T. Proebsting and J. Hartman, "Scout: A Communications-Oriented Operationg System," Department of Computer Science, The University of Arizona, no. 94-20, 1994.

[23] John Hartman, Larry Peterson, Andy Bavier, Peter Bigot, Patrick Bridges, Brady Montz, Rob Piltz, Todd Proebsting, and Oliver Spatscheck. " Joust: A Platform for Communications-Oriented Liquid Software ", IEEE Computer 32, 4, April 1999, 50-56.

[24] Quality of Service Filters for Multimedia Communications Nicholas Yeadon Ph.D. Thesis, Computing Department, Lancaster University, Bailrigg, Lancaster, LA1 4YR, U.K., May 1996. Internal report number MPG-96-35.

[25] M. Kadansky, D. Chiu, J. Wesley, J.Provino. "Tree-based Reliable Multicast (TRAM)" Work in progress, i-d - ftp from http://www.ietf.org/ietf draft-kadansky-tram-02.txt

[26] Fast Forward Network Inc. "Broadcast overlay architecture" http://www.ffnet.com/pdfs/boa-whitepaper.pdf

[27] RealAudio http://www.real.com/

[28] A. Baratloo, M. Karaul, H. Karl, and Z. M. Kedem, "An infrastructure for network computing with Java applets", In Proc. ACM Workshop on Java for High-Performanace Network Computing, 1998.

[29] Servlet `http://java.sun.com/products/servlet/index.html`

[30] Li Gong. "Inside Java 2 Platform Security". Sun Microsystems 1999 ISBN 0-201-31000-7

[31] FunnelWeb `http://dmir.socs.uts.edu.au/projects/alan/funnelWeb-2.0.1.html`

[32] Glen MacLarty, Michael Fry. "Policy-based Content Delivery: an Active Network Approach". The 5th International Web Caching and Content Delivery Workshop. Lisbon, Portugal, 22-24 May 2000.

[33] Duane Wessels, "Squid Internet Object Cache", Available on the World Wide Web at http://squid.nlanr.net/Squid/, May 1996.

[34] Bluetooth `http://www.bluetooth.com/`

[35] Y. Rekhter, T. Li. "An Architecture for IP Address Allocation with CIDR". RFC 1518 September 1993.