Ghosts of the Net!

IEEE Openarch 2001 – Short Paper Session
Anchorage, Alaska, April 28, 2001

http://www.openarch.org/

Call for Papers

Show us the new Ghosts of the Net: We want to know what these code spirits do there, how they look, how they behave, how you make them. Let the rays of mobile code penetrate the deepest corner of the dark network core and enlighten us with everything you know about the future of active networking, open signaling and programmable networks. Submit your venturesome but sound idea to the short paper session of Openarch'2001.

We are seeking short papers on novel concepts related to the Openarch'2001 themes. For additional guidelines, consult the open list below on what we would like and what we don't want to see:

Yes, please	*No, thanks*
Turing on the move	Jacquard in the node
Automate the net	Humans in the loop
Internet Deconstructivism	Internet plus epsilon
The fury in the net	Zero jitter architectures
Trust in code Darwinism	Believe in design
Kolmogorov	Lempel-Ziv
Code species	Standards mammoths
Self– ∗	Command and Control
Fragment, translate, tunnel!	You shall have only one address
Packet bartering	Bandwidth plan economy
. . .	. . .

Table of Contents

Foreword

Depending on what your reference points are, research in Active Networking is now more than 5, 10 or 20 years old. Did we do well in exploring this new territory? I don't think so! Classic networks are the dominant concern of most active, programmable networking and open signaling papers which gratefully produce fancy extentions and new features for the mighty Internet. Aah – the Internet! How suffocating it has become! Do we really need it as a target, as a reason, to do active networking research? I think this would be grossly underestimating the *power of mobile code* and the new *Ghosts of the Net*. The goal of this Short Paper Session's call for papers was to create a forum for concepts which reach beyond the existing network mammoths; and to encourage young researchers to go where the Internet cannot be.

The papers selected for this Short Paper Session span a broad spectrum, from the conceptual & theoretical to rather concrete engineering approaches. I hope that you will enjoy these presentations and that they encourage you to develop an even more radical active networking point of view.

I would like to thank the authors, including those whose papers we could not include in the program, for their contributions. I should also like to thank Andrew Campbell for his help in the selection of papers and the setting up of this event.

Christian F. Tschudin, Uppsala, April 2001

Active Network Management and Kolmogorov Complexity

Amit B. Kulkarni and Stephen F. Bush, GE Corporate R&D, Niskayuna, NY, USA
kulkarni@crd.ge.com, bushsf@crd.ge.com

This paper seeks to describe new and better ways to represent network health and thus attempts to explore concepts other than those based on network topology-based representations of network management. It examines the manner in which active network management can benefit from Algorithmic Information Theory. Due to the new paradigm and enhanced capabilities of active networks, this work proceeds along the lines that a new perspective that incorporates Algorithmic Information Theory can provide superior, innovative solutions for network management.

1.1 Introduction

Actives Networks [2, 5] enable the propagation of active packets, that is, network packets that carry executable code in addition to data. The executable code should be designed in a more compact form than transmission of the equivalent static, non-executable data in a piecemeal fashion. Clearly, the algorithmic nature of the active packet allows for more compression. As a simple example, a million digits of π can be transmitted, or more compactly and simply, the description of a circle and the command to divide the circumference by the diameter. This paper seeks to describe new and better ways to represent network health and thus attempts to explore concepts other than those based on network topology-based representations of network management. It examines the manner in which active network management can benefit from Algorithmic Information Theory. Due to the new paradigm and enhanced capabilities of active networks, this work proceeds along the lines that a new perspective that incorporates Algorithmic Information Theory [3] can provide superior, innovative solutions for network management.

The proposed approach uses Kolmogorov Complexity and the science of Algorithmic Information Theory (sometimes called Complexity Theory) to build self-managed networks and vulnerability analysis techniques that draw on fundamental properties of information to identify, analyze, and correct faults as well as security vulnerabilities in an information system. Bush [1] introduced the use of Streptichrons in the form of algorithmic information transfer in an active network. This paper proposes an approach for network management that uses complexity measures to detect and analyze problems in the network before applying self-composition techniques to remedy the problem. From an implementation perspective, these approaches will be deployed in an active network environment.

1.2 Proactive Network Management

The primary resources in a network are computing (CPU), memory, bandwidth and storage. Other critical elements of the network that contribute to the health of the network are the hardware interfaces (the network interface cards), the links between the network nodes, and the control software. Our premise for analyzing faults in the network is based on the notion that the shared resources have an “operating range” of behavior within which the network functions normally. Whenever one or more of these shared resources exceeds its normal range, the health of the network is said to be compromised. The irrational behavior of one of the resources can have a cascading effect on other resources, creating a ripple effect of problems that may mask the original problem. To quickly isolate and identify a problem, one has to understand the overall behavior of the network given different root causes. Observing the effects of root causes on the operating ranges of the shared resources enables rapid identification of a problem. Understanding emergent behavior induced by root causes enables us to design and implement remedies that recognize emergent behavior and understand the root cause and implement solutions to alleviate the problem. In fact, if sufficiently distinguishing emergent behavior exists, problems can be identified in very early stages of their manifestation.

The objective of this work is a step towards communication networks whose inherent state, or natural tendency, is optimum performance. Faults should naturally attract the entities required to eliminate faults. The proposed mechanism to accomplish the attraction of solution entities can be summarized in three steps. The first step is detecting the problem. Current approaches use reactive methods in which devices in the network notify a management station of its current status. Specialized software at the management station sifts through all the data and determines if a fault has occurred. If a fault has indeed occurred or if certain Management Information Base (MIB) [4] values appear out-of-range, an alarm is raised and the network manager is notified. This approach is reactive and cannot respond quickly to faults. On the other hand, we utilize a proactive approach in which the fault must identify itself to the entities capable of eliminating it. Notice that this does not necessarily require a human to identify or understand the fault, only that the solution components of the management application are capable of recognizing it. Once the fault is identified, the next step requires that the information be efficiently and accurately propagated to the solution entities with minimal overhead to the network. Once a fault occurs, the network may already be impaired; adding additional overhead in trying to solve the problem would only exacerbate the problem. To re-state this step from another viewpoint, the necessary and sufficient solution entities should be attracted to the proper locations to solve the problem. The final step in this vision is that only the necessary and sufficient entities required to correct the fault arrive, and that these solution entities act in a co-operative manner to correct the fault quickly and accurately. In this paper, we describe techniques and means to achieve automated fault identification.

1.3 Complexity and Network Health

According to Complexity Theory, the complexity of a piece of information is the size of the smallest program capable of producing that piece of information. There are strong ties between Algorithmic Information Theory and Complexity. A truly random piece of information cannot be compressed, and its length is its complexity. Complexity is, in general, uncomputable. However, bounds on complexity can be derived, unfortunately page limitations do not allow us to discuss this aspect in more detail.

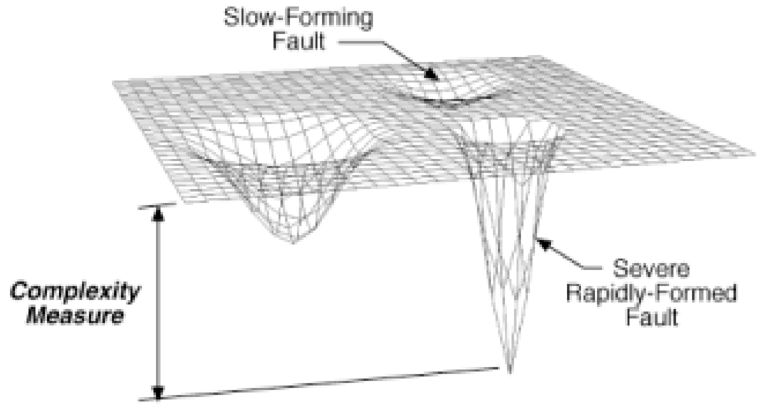

Figure 1.1: Complexity-Based View of Network Health.

Complexity-based approach to network management provides certain advantages over a topology-based representation. A complexity-based paradigm can exploit relationships between MIB variables on different branches of the MIB tree and those that are spatially scattered in the network. These relationships can be used to compute complexities of the network in different dimensions. These different dimensions could relate to availability of network services, health of devices, components and sub-components in the network, performance of user applications, vulnerability analysis, and other relevant network services and applications. Applying results from Algorithmic Information Theory, one can compute the complexities of the various dimensions. We use the assumption that greater complexity implies more randomness, which in turn implies that the system is healthy. Regions of low complexity indicate a problem that has occurred or may occur. The measure of complexity indicates the severity of the problem. Linking problem severity to complexity measures has further advantages. This enables mechanical quantification of problems occurring in the network, enabling development of sophisticated management applications that can enable self-healing within the network. Furthermore, the slope of the complexity curve can be monitored to indicate future problem occurrences. When a region of complexity in a certain dimension changes its measure from high to low, it indicates that the related network component is likely to develop a fault as illustrated in Figure 1.1.

1.4 Complexity Model and Experimental Validation

To visualize the concept of complexity-based representation of network health, imagine a space filled with entities that represent the values of various monitored objects from the managed system. For instance, each entity could be an SNMP MIB Object and its corresponding value. Initially, all entities are randomly located in that space. Furthermore, each entity can move in a random fashion in an area around its initial location. By complexity theory, a truly random sequence is incompressible; therefore, the sequence representing the location of these entities cannot be compressed.

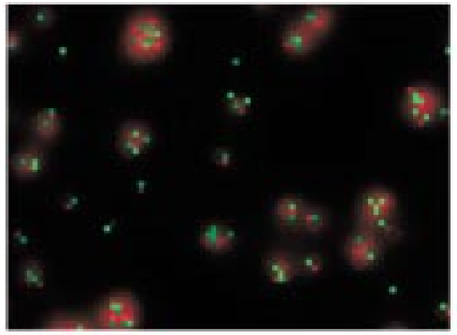

Figure 1.2: Nebulae of MIB Objects Formed from Fault (Heat).

Each entity has a normal operating range within which its value should fall during ”normal” operation. As the operating range is exceeded, heat is generated as shown in Figure 1.2. We use the characteristics of heat diffusion provided by the Swarm library package to represent dissemination of the information about the network service or component in the network. As heat diffuses into space its energy is dissipated. Similarly, information dissipates upon the passage of time finally making it obsolete. Other entities are attracted to the heat, forming circular patterns, or clusters. This represents the formation of relationships between the different entities that are affected by the problem.

Figures 1.3 and 1.4 show the entities and the heat generated from the entities after one hundred simulated time units. The pattern formation results in a loss of randomness and enables a compressed representation of the problem. As the magnitude and number of heat clusters increases, randomness decreases and compressibility increases. The result is that more severe faults can be represented in smaller and more efficient forms for transmission. Thus, a relationship between network health and complexity is established. Figures 1.3 and 1.4 show the cluster rate for a simulation with a specified probability of fault occurrence, duration, and severity.

In other words, when a fault occurs or can potentially occur, relationships between the various components are exposed due to the information transfer (causing the clusters to coagulate and become dense). This in turn enables information to be transmitted algorithmically (that is, as relationships and data) instead of long sequences of raw data.

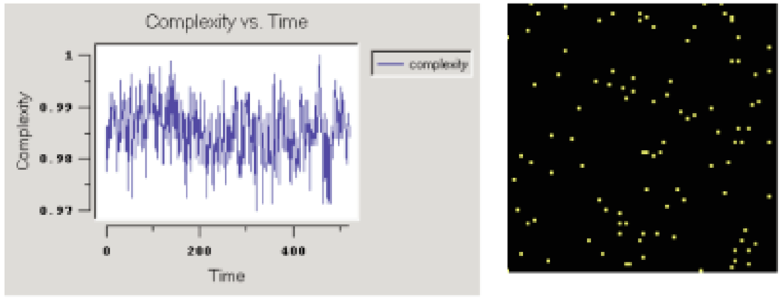

Figure 1.3: Healthy Network.

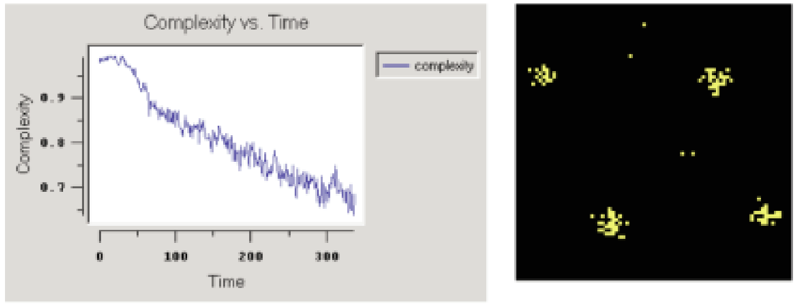

Figure 1.4: Multiple Faults.

One of the goals of this work is to pursue better complexity metrics. Measuring clustering is an attempt to measure complexity. We define the cluster measure to be one minus the proportion of the entity that is surrounded by adjacent entities. Note in Figure 1.4 that clustering clearly occurs. Each entity in this experiment had the same probability of fault occurrence and severity.

1.5 Summary

Proactive network management is an area that has not received much attention in networking research circles. The ideas described in this paper outline a novel approach for an active network that inherently attracts solutions to network problems, even before such problems occur. Proactive management will allow the creation of networks that are robust, as well as resistant to security threats such as intrusion and denial of service. This paper takes the first step in applying algorithmic information theory and techniques to network management to create the next generation of networks that are self-diagnosing, self-managing, and self-healing.

References

[1] Stephen F. Bush. Active Virtual Network Management Prediction. In *Parallel and Discrete Event Simulation Conference (PADS) '99*, May 1999.

[2] Stephen F. Bush and Amit Kulkarni. *Active Networks and Active Virtual Network Management*. ISBN 0-306-46560-4, Kluwer Academic/Plenum Publishers, Boston, March 2001.

[3] Ming Li and Paul Vitanyi. *Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, August 1993.

[4] Marshall T. Rose. *The Simple Book, An Introduction to the Management of TCP/IP-Based Internets*. Prentice Hall, 1991.

[5] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

Self-Organizing Route Aggregation for Active Ad-Hoc Networks

Richard Gold, GMD FOKUS
gold@fokus.gmd.de

We describe a self-organizing approach for providing route aggregation for Active Ad-Hoc Networks. Ad-Hoc networks are defined as ‘infrastructure-less networks’. This extreme approach to decentralization leads to many problems in scalability. In order to combat this problem we propose a solution which introduces the notion of hierarchy and route aggregation for Ad-Hoc networks, in a similar way that BGP did for the Internet. Our solution uses application-layer Active Networking as a base technology so that the overlay network we create can then tune itself to current networking conditions.

2.1 Introduction

We describe a self-organizing approach for providing route aggregation for Active Ad-Hoc Networks. Ad-Hoc networks are defined as 'infrastructure-less networks' and are the topic of discussion of various groups, including the IETF's MANET working group [1]. The MANET group mainly deals with wireless Ad-Hoc networks, however, the principles behind these Ad-Hoc networks are also used by peer-to-peer systems such as Gnutella [2] and FreeNet [3]. These peer-to-peer systems take a radically decentralized approach to the network infrastructure and remove the distinction between dedicated client and server devices. In an Ad-Hoc network all nodes function as clients and as servers. Also in an Ad-Hoc network, all nodes must function as routers, as some nodes are only reachable by multi-hop paths due to the lack of complete network reachability or route knowledge by all nodes.

It is the problem of routing in Ad-Hoc networks as typified by peer-to-peer systems that we deal with in this paper. Gnutella, for example, is a completely decentralized system where, in order for a user to locate a particular file, it must perform a broadcast search to all other Gnutella nodes. This "brute-force" approach to decentralization leads to many problems in scalability, as reported in ZDNet [4]. In order to combat this problem we propose a solution which introduces the notion of hierarchy and route aggregation for Ad-Hoc networks, in a similar way that BGP did for the Internet [5]. Our solution uses application-layer Active Networking as a base technology as we would like our peer-to-peer application's routing policy base to be programmable, so that the application-layer overlay network can then tune itself to current networking conditions, thus improving the robustness of the system. This is similar to work done in the ARRCANE project at Uppsala University [6], but they focus on the deployment of active routing protocols at layer 2.5, whereas our approach is concerned with the aggregation of routes provided by the protocols at the application layer.

2.2 Ad-Hoc Overlay Network Design

2.2.1 Definitions

We imagine the nodes of our network to be capable of performing some or all of the following functions, depending on their capabilities:

Normal operation : This function constitutes packet forwarding on behalf of other nodes in the overlay network. This is the default behaviour for nodes in the network.

Helper node function : If a node performs this function then when a new node wishes to join the overlay network, the helper node can provide information about other nodes in the network.

Aggregation point : This function is equivalent to the role of a BGP speaker in the traditional Internet. That is to say that the aggregation point advertises reachability information to other aggregation points in the overlay network.

We also define the nodes of the network to be running an Active Network platform on top of which the application of our system is implemented on. The routing tables of our application are thereby programmable by the Active Network. Our network is an application layer overlay network; the nodes in the network form their own overlay network which is application specific.

2.2.2 Nodes joining & leaving the Ad-Hoc Overlay Network

In the beginning when a node comes online it can register with a helper node. The names of the helper nodes are obtainable through out-of-band communication channels such as websites, magazine publications etc., much in the same way that Gnutella or Napigator addresses are distributed. Note that the helper node is not a single point of failure as it is merely one of the many nodes of the network which are performing this function. The helper node (e.g., `gnutellahosts.com:6346`) then transfers a history list of all other known aggregation points to the new node. This is performed by using the Active Network to transfer code to program the new node's routing tables to the reflect the new configuration. The new node then makes a measurement to the aggregation points in the history list using metrics like hop count, delay etc. - this is similar to the techniques used by the SOAR architecture [7]. This is performed by Active Network measurement code so we can perform computation on the overlay network nodes on the path to the destination aggregation point. The decision could also be made by a policy-decision as a node may have routes that it prefers due to some routing analysis code executed by the user, for example a Detour route [8]. Based upon this decision the new node joins one of the aggregation groups. Group assignment involves two steps:

Firstly, the attribution of an application number to the node (these are addresses allocated solely for the overlay network, similar to the way the ABONE [9] allocates addresses for its overlay network). Their structure is shared with the IPv4 address format (IPv6 could also be a possible candidate), although these application-layer addresses have no relation to the network layer. The numbers allocated by the system are suitable for aggregation, as each Autonomous System (AS) in the network has its own block of addresses to allocated to new nodes. An AS in our system is defined as a group of nodes whose application numbers can be aggregated.

Secondly, the transferal of the tuple `<application number & IP address>` of all the aggregation points in the network to the new node. This is similar to the history list used in the WormNet from the Samhain project [10].

When a node then quits gracefully from the network it informs its aggregation point by means of an active packet, which can then re-adjust its routing table advertisements. When an aggregation point suddenly disappears from the network, the members of its group call an election for a new aggregation and broadcasts its routing information to the other nodes in its former AS. Such an election could be performed using the negotiation protocol described in [11]. A virtual currency system for Ad-Hoc Networks like Nuglets [12] could be used to encourage nodes to occasionally to become aggregation points, rather than just making it the default behaviour that one randomly selected node gets elected. Such a virtual currency system could also be used to ensure that there are always enough helper nodes and aggregation points in the network by making the amount of currency a node gets for assuming a certain duty inversely proportional to the amount of nodes performing that function in the network. When an aggregation point disappears unexpectedly from the network, the nodes that the aggregation point was responsible for can consult their history lists to discover another aggregation point to connect to if the above election technique fails to work.

2.3 Hierarchy and Aggregation in the Ad-Hoc Overlay Network

In order to achieve our goal of aggregation in the network, there must be a notion of hierarchy so that nodes in one level of the hierarchy do not necessarily have to be visible to nodes in a higher level of the hierarchy. The hierarchy in our system is imposed through the naming service in the same way as BGP [5] enforces hierarchy where networks are advertised through their backbone providers.

2.3.1 Naming Service & Routing

Each aggregation point in the network maintains a list of the tuples `<application number & IP address>` of its AS. The aggregation points then dispatch active packets which, when executed, will update the routing tables on the destination aggregation points to reflect which network numbers the source aggregation point is advertising. The network numbers are the aggregation of the application numbers of the ASes that they represent. This process is shown in Figure 2.1.

When a connection between two nodes in the network needs to be established, then the application number is looked by a node querying its aggregation point. The application number is then used to route the packet through the overlay network to the destination aggregation point which then uses the actual IP address of the destination node to internally route the packet.

2.3.2 Aggregation

The aggregation mechanism we propose works in the following manner: each AS in the network has its own address range with which it allocates application numbers to new nodes that join the network. E.g. AS-FOKUS could have the range 192.168.1.x where x is a number between 1 and 255. Thus, in the inter-AS routing tables (akin to BGP), each group can be represented by one entry e.g., AS-FOKUS is represented as 192.168.1. When a new node joins AS-FOKUS it is assigned an application number. A node could also have application name associated with it, for example the username from the application that is being used (E.g., Napster username) to assist with information location in a content-based routing environment such as the TRIAD project [13].

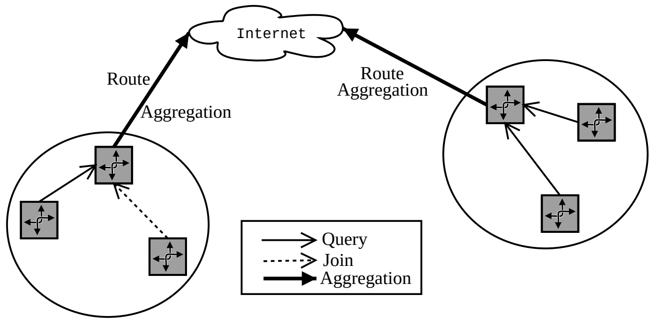

Figure 2.1: Route Aggregation exchange between ASes

The aggregation points in the network evaluate the amount of nodes in the routing tables they exchange. When the amount of nodes in an AS falls below a certain threshold (configurable at runtime, although a default could be included), then the aggregation point can ask its parent if its group can be coalesced with the group of its parent. In order for the groups to be coalesced, the requesting aggregation point transfers its routing information with active packets to the new aggregation point and can then unload its Active Network routing code.

When an aggregation point's address space exceeds a certain threshold of nodes, it will try subnet its current application number and spawn a new instance of the aggregation point Active Network code onto an elected node, using the negotiation protocol described earlier, which will then also assume aggregation point responsibilities. See Figure 2.2. The new instance is then allocated an address space range by the node which spawned that instance. Another option is that the node can coalesce with its parent and ask its parent to subnet on its behalf in the case when its own address space threshold is reached. This way the child merges its address space range with its parent. The rationale behind this on-demand technique is that we wish to keep the node software as lean as possible, and only have the aggregation point code on-demand. This may lead to problems with bootstrapping in situations where no reachable nodes have the aggregation point code - this is an area of future work.

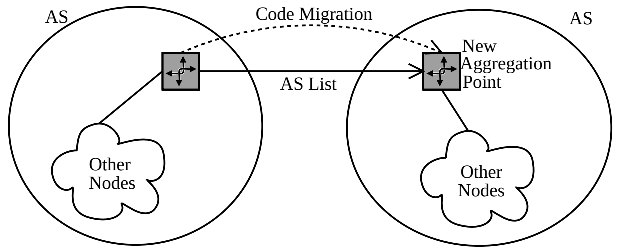

Figure 2.2: Spawning of a new aggregation point

2.4 Usage of the Active Network

In our implementation scenario, the application would be a peer-to-peer file sharing application, such as Gnutella [2] or FreeNet [3], implemented on top of an application-layer Active Network platform. When an application accesses a helper node for the first time, it receives an active packet containing the current routing and policy information for the AS that the node is participating in. This code then executes on the Active Network platform in order to set up and manage the overlay network.

When a new aggregation point needs to be created, the code running on the Active Network platform actually implements this functionality by spawning itself onto the newly elected node. This on-demand approach allows for the flexibly and dynamic upgrading and modification of the application-layer routing infrastructure. For example, if a user would like to introduce a new policy decision mechanism for deciding a Detour route, then this code just has to be injected into node in order to immediately start assisting with the routing decisions. Another advantage is that we only run the aggregation point code on nodes that actually are aggregation points.

The speculative future of such an approach is to isolate the various networking components of applications and to allow the mixing and matching of the genetic code of, for example, a peer-to-peer file-sharing application and various application-layer routing protocol efforts. Thus when a peer-to-peer file-sharing application needs a decentralized, self-organizing routing infrastructure it can just access various alternatives from the Active Network which are able just to plug in to an existing application.

2.5 Conclusions

Whilst there are systems that provide self-organizing routing in overlay networks [7] and systems that provide hierarchical routing in an Ad-Hoc wireless network [14], there are none as yet that actually provide self-organizing route aggregation for Ad-Hoc overlay networks. Route aggregation is crucial in the current Internet for scalability reasons, but for an Ad-Hoc network the current scheme is too fixed as it relies on the static definition of an AS' boundaries. The explicit introduction of self-organization as a method for maintaining a functioning route aggregation infrastructure is important in order to deal with the unpredictable nature of Ad-Hoc networks. It enables the system to autonomously scale the size and layers of its hierarchy up and down. We propose an implementation based on application-layer overlay networks and Active Networks. We use an overlay network to allow scalability through route aggregation on the application-layer. The Active Network is used to allow the dynamic changing of the application-layer routing infrastructure and the runtime deployment of infrastructure code onto nodes

Such a system would be useful for peer-to-peer networks [3], Ad-Hoc wireless networks [15] and Active Network testbeds such as [9] as it allows the introduction of route aggregation for systems which would typically otherwise suffer from scalability problems.

The current status of the work is that the architecture is finished and implementation work will commence in the first quarter of 2001.

References

[1] IETF MANET Working Group. MANET Charter, 2000. `http://www.ietf.org/html.charters/manet-charter.html`.

[2] Justin Frankel and Tom Pepper. The Gnutella Project. `http://gnutella.wego.com/`, 2000.

[3] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. FreeNet: A distributed anonymous information storage and retrieval system. *Workshop on Design Issues in Anonymity and Unobservability*, 2000. `http://www.freenetproject.org/icsi-revised.ps.gz`.

[4] Aaron Pava. Gnutella is dead. *ZDNet Music*, 2000. `http://music.zdnet.com/features/highnote/092100_gnutella_dead.html`.

[5] K. Lougheed and Y. Rekhter. Request For Comments 1163: A Border Gateway Protocol, 1990. `http://gatekeeper.dec.com/pub/net/info/RFC/rfc1163.txt`.

[6] Christian Tschudin, Henrik Lundgren, and Henrik Gulbrandsen. Active Routing for Ad-Hoc Networks. *IEEE Communications Magazine*, pages 122–127, 2000.

[7] Atanu Ghosh, Michael Fry, and Jon Crowcroft. An architecture for application layer routing. In *Second International Working Conference on Active Networks IWAN 2000*, 2000.

[8] Savage, Anderson, Aggarwal, Becker, Cardwell, Collins, Hoffman, Snell, Vahdat, Voelker, and Zahorjan. Detour: a case for informed internet routing and transport. *IEEE Micro*, pages 50–59, 1999. `http://www.cs.washington.edu/homes/savage/papers/IEEEMicro99.pdf`.

[9] Steve Berson. A gentle introduction to the ABone. *OPENSIG Workshop*, 2000. `http://www.isi.edu/abone/DOCUMENTS/opensig00.berson.ps.Z`.

[10] Michal Zalewski. ”I don't think I really love you”. `http://lcamtuf.na.export.pl/worm.txt`, 2000.

[11] Richard Gold and Dan Tidhar. Concurrent routing protocols in an Active Ad-Hoc Network. In *AISB Symposium on software mobility and adaptive behaviour*, 2001. `ftp://ftp.fokus.gmd.de/pub/glone/usr/rgo/pub/AISB01.ps.gz`.

[12] J. P. Hubaux, Th. Gross, J. Y. Le Boudec, and M. Vetterli. Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks. Technical report, Swiss Federal Institute of Technology Lausanne, 2001. `http://www.terminodes.org/publications/dsc2001001.ps`.

[13] Mark Gritter and David Cheriton. An architecture for content routing support in the Internet. In *USITS 2001*, 2001. `http://dsg.stanford.edu/triad/usits.ps.gz`.

[14] Josh Broch, David A. Maltz, and David B. Johnson. Supporting hierarchy and heterogeneous interfaces in multi-hop wireless ad hoc networks. In *Proceedings of the Workshop on Mobile Computing*, 1999. `http://www.monarch.cs.cmu.edu/monarch-papers/ispan99.ps`.

[15] J. P. Hubaux, Th. Gross, J. Y. Le Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: the terminodes project. *IEEE Communications Magazine*, 2001. `http://www.terminodes.org/publications/commag01a.pdf`.

ALM and ProgNets for v4-to-v6 Multicast Transition

Paul Smith, Laurent Mathy, Lancaster University, UK
p.smith@comp.lancs.ac.uk, laurent@comp.lancs.ac.uk
Roberto Canonico, Università di Napoli, Italy
roberto.canonico@unina.it
David Hutchison, Lancaster University, UK
d.hutchison@lancaster.ac.uk

Current efforts in the area of IP transitioning have largely considered unicast communications. A number of challenges that are associated with the IP multicast transition make the problem distinct from the unicast scenario. Application Level Multicast (ALM) is a technique that can be used to enable the operation of multicast services in non-native multicast environments. In this short paper we propose a combined use of ALM and programmable networking technologies to speed up the IP multicast deployment in an IPv4 to IPv6 transitioning Internet.

3.1 Introduction

The introduction of IPv6 in the Internet requires a change in the routing components of the network infrastructure. Since an instantaneous upgrade from IPv4 to IPv6 is neither possible nor desirable, viable transitioning mechanisms are essential to the evolution of the Internet. These mechanisms must be efficient, so the two protocols can coexist in the Internet for an interim period. Current efforts in the area of IP transitioning have largely considered unicast communications. A number of challenges that are associated with the IP multicast transition make the problem distinct from the unicast scenario. A classical technique for incremental deployment of network-layer protocols is based on gradual deployment of the new protocol in isolated network "islands" and subsequent interconnection of islands by means of tunneling. However, since tunnel end-points are manually and statically established, such an approach leads to inefficient overlaid networks. Application Level Multicast (ALM) is a means of enabling service provision in non-native multicast environments. ALM enables multicast-style communication to be conducted using only unicast messaging between parties in an ALM spanning tree. Programmable networking technologies [1], promise to enable flexible and arbitrary service provision, by providing open programmable interfaces. In this paper, we suggest to combine the classical tunneling mechanism with an efficient and scalable Tree Building Control Protocol running in programmable edge devices for automatic establishment of tunnels. This approach can speed up multicast deployment in an IPv4 to IPv6 transitioning Internet.

3.2 Application Level Multicast

Application Level Multicast (ALM) is a means of providing multicast-style communication using only unicast infrastructure. This is achieved by building a spanning tree between the hosts which aim to perform this multicast-style of communication. The spanning tree is then used to tunnel data between the nodes on the tree.

3.2.1 Constructing ALM spanning trees

At Lancaster University, an approach to building ALM spanning trees is under development [2]. Such an approach is based on a distributed algorithm which requires only partial knowledge of group membership and relies on local decisions to build a spanning tree among the members of a multicast group. The algorithm for constructing the tree can be summarised as the following:

1. Nominate an appropriate node as the root. The choice of root node will be largely dictated by the specific application of the spanning tree. For example, if only a single source exists, the obvious choice of root node would be the source.

2. Nodes wishing to join the ALM spanning tree, contact the root node in order to determine their place in the tree. The nodes place in the tree is determined by the result of a score function. The application of the spanning tree will dictate the parameters to the score function. For example, if the tree is to be used for control purposes (signaling), obvious parameters to the score function would be the round-trip-time between nodes and a measure of packet loss. It makes no sense to consider bandwidth as a score function parameter for example, as control traffic does not have significant bandwidth requirements.

3. Based upon the result of the score function, the node can either then become one of the root's children, or can be sent to one of the root node's current children, where the score function is calculated again. This process is performed recursively until the spanning tree falls into a stable state. Each node in the tree has local knowledge regarding the number of children it may wish to support. If it is appropriate for the new node to become a child of a given parent, and that parent would then exceed their child limit by accepting this new child, the position of a current child must be reassessed. In this case, the child node to be moved is then sent to one of the parents current children (or the new one) where its new position is determined based on the score function.

To realise the above distributed algorithm a suitable Tree Building Control Protocol has been designed and is being implemented at Lancaster University. Other notable approaches to building ALM spanning trees include [3] [4].

3.2.2 Algorithm properties

Adopting the approach outlined in section 3.2.1 to building ALM spanning trees has a number of beneficial properties. Using this approach, only partial knowledge of the tree is required. This partial knowledge consists of the root and your children. This has the property of making the algorithms for constructing the tree relatively simple and therefore the nodes less complex. Furthermore, as only partial knowledge is required for constructing the spanning tree in this manner, the algorithm should scale significantly better than if global knowledge is required.

Additionally, algorithms have been developed to enable the tree to be reshaped dynamically. For example, people can leave and join the tree arbitrarily during the spanning tree's lifecycle. This property also enables the tree to be optimised. ALM being used to repair partitioned network level multicast trees, is an example where such optimisation may take place. In this scenario, the primary object would be to construct the ALM tree between the partitioned regions rapidly, thus enabling the continuation of the multicast session. This initial spanning tree may well be suboptimal, if so, the tree could then be reshaped and optimised for the application using appropriate parameters to the score function.

No specialised infrastructure is required for the spanning tree construction approach outlined in section 3.2.1 for ALM to operate. The construction is conducted between nodes that are party to the ALM communication, typically end-systems, and therefore existing protocols can be used to transmit any traffic necessary between them.

3.3 Multicast in an IPv4 to IPv6 Transitioning Internet

There are efforts being made in the Internet community toward the migration from the current ubiquitous IP version (IPv4), to the proposed next large-scale deployed version of IP (IPv6). Migration from IPv4 to IPv6 should take place over a period of time, during which the two protocol versions must co-exist in the Internet. Several approaches to undertaking this transition have been proposed [5]. These approaches have largely considered point-to-point communication, using statically configured tunnels between nodes or routers. In a multicast environment, providing transitional mechanisms is more problematic for several reasons. In particular, the classical mechanism based on static tunnels is challenged by the fact that, in the case of multicast, the source of the multicast traffic is unaware of the location of receivers. The problem is even more difficult in the case of multiple sources transmitting to the same multicast group. Constructing tunnels between an arbitrary number of dynamically available sources and destinations is inefficient and unscalable, as demonstrated by the MBone.

3.3.1 Our approach to multicast migration

Consider a scenario, in which a number of users in different IPv6 multicast island, wish to conduct a multicast session over an IPv4 backbone, as depicted in figure 3.1a. This session would fail to operate as the islands are disconnected by the IPv4 backbone. An approach to solving this problem should have the property of being transparent to the nodes in the multicast islands, resulting in no changes in the end-systems. Additionally, nodes should be able to dynamically join and leave the multicast session, as is normally the case. An approach should also scale sufficiently and not introduce exponential levels of complexity into devices.

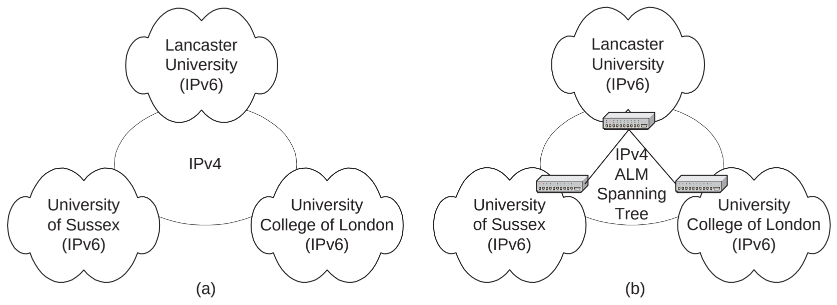
Lancaster University (IPv6)
IPv4
University of Sussex (IPv6)
University College of London (IPv6)
(a)
Lancaster University (IPv6)
IPv4 ALM Spanning Tree
University of Sussex (IPv6)
University College of London (IPv6)
(b)

Figure 3.1: a) IPv6 islands that wish to conduct a multicast session, however an IPv4 backbone inhibits traffic from reaching the disconnected islands; b) by constructing an ALM spanning tree using programmable devices, the IPv6 multicast islands can communicate.

We propose an approach to resolving this issue based upon ALM that leverages programmable networking technologies. Programmable devices in their first generation are anticipated to reside at network edges [6], as shown in figure 3.1b. Such a programmable network infrastructure can be used by ALM to deploy on-demand, dynamic tunnels, resulting in highly efficient overlay multicast services.

The solution we propose involves constructing an ALM spanning tree between programmable devices interconnecting the IPv6 multicast islands, as well as possibly individual hosts, taking part in a multicast session. These devices can then encapsulate the IPv6 multicast packets being sent in IPv4 unicast packets and disseminate them to appropriate nodes on the ALM spanning tree. Likewise, the programmable devices can de-encapsulate the IPv6 multicast traffic and forward it toward local nodes that have registered for the traffic. It should be noted that an IPv6 multicast island without a programmable edge device can always be connected to the programmable network infrastructure via a pre-configured tunnel.

In using the approach to constructing ALM trees developed at Lancaster University based on programmable devices, IPv6 nodes can dynamically join and leave the multicast group. With the use of programmable devices taking the role of a local multicast router, the operation of this solution is transparent to end-stations. Connecting the multicast islands using ALM also simplifies the operation of the programmable devices as they interact using a generic protocol (ALM) between potentially heterogeneous multicast routing protocol domains. This approach should scale well, with the use of our technique for constructing ALM trees, which requires only partial knowledge of the tree, and therefore programmable devices will not have to maintain large amounts of state irrelevant to their operation.

3.4 Conclusion and Further Work

The Internet is evolving constantly: one such evolution is that of the Internet Protocol. Current approaches to providing the transition from IPv4 to IPv6 largely only consider unicast communication. We have presented an approach to providing IP multicast transition using ALM and programmable networking technologies. This approach enables the transparent connection of IPv6 multicast islands that are disconnected by IPv4 network regions. This approach enables the dynamic reconfiguration of the multicast session, by inheriting properties from an approach to building ALM spanning trees based on programmable networks. Furthermore this approach will scale to a potentially large number of disconnected multicast islands.

Finally, future work will also study the use of ALM over a programmable network infrastructure to enable the interoperability of mixed IPv4 and IPv6 multicast sessions, possibly interconnecting islands using different multicast routing protocols.

References

[1] A.T. Campbell, M.E. Kounavis, J.B. Vicente, D. Villela, K. Miki, and H.G. De Meer. A Survey of Programmable Networks. In *ACM SIGCOMM Computer Communication Review*, April 1999.

[2] R. Canonico, L. Mathy, and D. Hutchison. A Transport-Level Protocol Suite for Multimedia Multicast Communication over an SSM-enabled Internet. In *NGC 2000 poster session*, Stanford University, Palo Alto, California, USA, November 2000.

[3] Y. Chu, S.G. Rao, and H. Zhang. A Case for End System Multicast. In *ACM Sigmetrics 2000*, Santa Clara, California, USA, 2000.

[4] P. Francis. Yallcast: Extending the Internet Multicast Architecture. Technical report, NTT Information Sharing Platform Laboratories, September 1999. http://www.yallcast.com/.

[5] R. Gilligan and E. Nordmark. Transition Mechanisms for IPv6 Hosts and Routers. RFC 1933, IETF, April 1996.

[6] I. Marshall, S. Covaci, T. Velte, A. Juhola, S. Parkkila, and M. Donohoe. The Impact of Active Networks on Established Network Operators. In *IWAN 99*, Berlin, 1999.

EdgeMeter: Distributed Network Metering

Marcelo Pias, Steve Wilbur, University College London (UCL)
m.pias@cs.ucl.ac.uk, s.wilbur@cs.ucl.ac.uk

In this short paper we give an overview of work in progress to propose a distributed metering model using Active Network concepts. The business model which motivates this work poses accounting requirements between different kind of providers and customers. However, security issues also need to be addressed. We briefly describe the problems being investigated which relate to the business model used. We have split the problem mainly into four subtopics. They are concerned with questions of how to control the system through high level policies, how to translate from such policies (Tariff/SLA) to a low level device dependent configuration. Other questions are concerned with what threats a system like this may be exposed to and what the limitations are in terms of system performance when measuring network traffic for QoS monitoring, billing and network planning.

4.1 Introduction

Our intention is to provide a means for building a safe distributed meter system for QoS monitoring in IP networks. The model is dynamically controlled by high level policies such as charging tariffs and service level agreements (SLAs). The Meter system has the intelligence for deciding the data needed, when should be forwarded and for whom.

In [1] the author describes another approach for IP accounting using Active Networks with accounting functions performed by the Provider's system only. Unlike in [1], our model not only allows accounting processes to be performed at provider's system but also at customer's premises. Furthermore, as stated before we are very concerned with the security and performance issues imposed by the creation and operation of the system.

4.2 Business Model

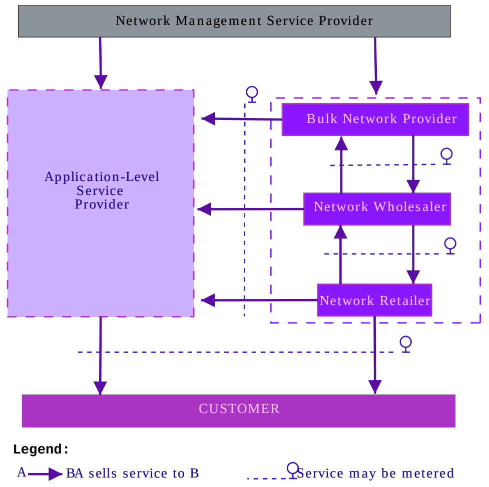
Network Management Service Provider
Application-Level Service Provider
Bulk Network Provider
Network Wholesaler
Network Retailer
CUSTOMER
Legend:
A → B A sells service to B
Service may be metered

Figure 4.1: Business Model

According to the legend in Fig. 4.1, some interfaces between the roles may be metered. Therefore, the Meter system may be physically placed between Providers boundaries for bulk measurements. On the other hand, meters can be installed between Providers (Network Retailers or Application-Level Service Provider) and Customers for fine-grained measurements.

4.3 Distributed Meter Model Proposal

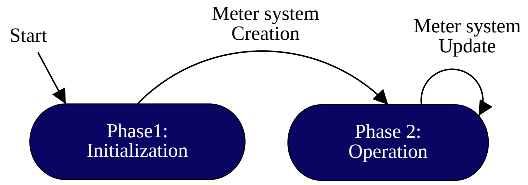

Figure 4.2: Distributed Meter Model

Our model proposal is composed by two phases (states) and two events. The event of *Meter Creation* moves the system from initialization to operation. A *Meter system Update* event occurs when the system demands any software/hardware update. Basically, there is a process of cutting over to the new code associated to this latter event.

We have modeled the system using some of Active Network (AN) concepts. In the paper [2], the authors propose an architecture for AN at application layer. Additionally, some implementation of such a model is described in [3]. We have concentrated much more on Application Level Active Networking (ALAN) model described in those references.

4.3.1 Definitions

- **Proxylet:** third parties' mobile code to be executed into a Generic Active Node (GAN). Proxylets are loaded by reference to Proxylet Repository Servers (PS) [2].
- **Generic Active Node (GAN):** it receives proxylets for execution on behalf of third parties. In fact, any type of proxylet is allowed to execute into a GAN.
- **Specialized Active Node (SAN):** in certain cases might be useful to specialize a GAN through filters which accepts specific group of proxylets to be executed.
- **Meter System:** a Meter system consists of a set of Meters plus a set of Dynamic Accounting Controllers (DAC). One of the features of the Meter system is the capability of QoS measurements.
- **Meter:** specialized active node created when a GAN receives a Meter_Code proxylet. Thus, the resulted node is called only Meter. In general, the Provider

should control the Meter through specific configuration (policies). A Meter captures packets from the network and apply policies in order to create a table of measurements.

- **Dynamic Accounting Controller (DAC):** another specialized active node created upon a GAN has received a Dynamic Accounting Controller (DAC) proxylet. This SAN forms the second part of a Meter system. Its main responsibilities include coordinating set of Meters, collecting meter data and transforming it into accounting data.
- **Tariff/SLA Translator (TT):** proxylet responsible for compiling the Tariff/SLA specification into Meter and Accounting policies used to instruct the Meter system.

4.3.2 Model Phase 1: Initialization

In this phase, the main components of the Meter system (Meter + DAC) are sent as mobile code over the network (Fig. 4.3)

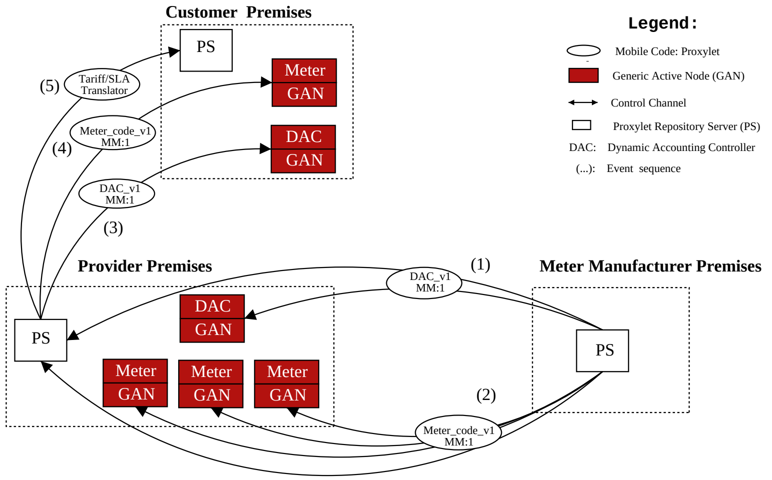

Figure 4.3: Phase 1 (Initialization)

Initially, the DAC proxylet version 1 is delivered to the Provider premises (1). The Meter_Code version 1 is transmitted to the Provider's Generic Active Nodes. A copy is stored at local Proxylet Server at Provider's premises (2). In steps 3, 4 and 5 the Meter, DAC and Tariff/SLA Translator proxylets are shipped to Customer's premises. They carry version numbers and Meter Manufacturer (MM) identifiers. Those are useful for the correct management of such proxylets.

4.3.3 Model Phase 2: Operation

In step (1), Tariff and Tariff Translator proxylets are loaded into Dynamic Accounting Controller (DAC). The Provider sends Tariff proxylets to his Customers, either through multicast or unicast communication (2). Once a Customer's DAC has received it, a Tariff/SLA Translator proxylet is loaded in order to perform the translation into appropriate Meter/Accounting policies (3). They specify explicitly requirements in a way that the Customer's Meter system can cope with. The DAC collects meter data either using push or pull modes(4). The Customer's DAC sends accounting data (detailed or summarized) to Provider's DAC. Finally, such collected data may be used for Billing, Network planning, QoS monitoring applications.

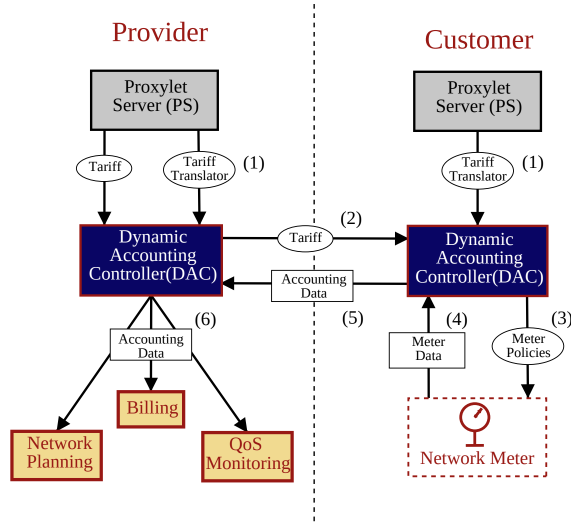

Figure 4.4: Phase 2 (Operation)

4.4 Security Requirements

We have so far identified the requirements below for the model explained in the previous section. Some of them are also discussed in [4] [5] [6].

- **(A) Execution of Safe Code(Meter Code,DAC,Tariff)**

  This would assure the integrity of the system where the proxylet is executed. Common problems to prevent are the installation of any malicious code which may compromise the role's system. This requirement is even strong when the Meter system is located at Customer's premises.

- **(B) Metering and Accounting Policies Soundness**

  This would make guarantees for Customers of the soundness property of meter and accounting policies (meter system configuration). The meter may not be tampered with by any of the roles in the model. When the meter system configuration is proved to measure safely what the actual tariff/SLA says, this requirement will be matched.

- **(C) Mutual Authentication**

  Mutual authentication between roles is essential. A role needs to know precisely who the other end of the communication is. Thus, they should identify themselves with some unique credentials.

- **(D) Confidentiality/Privacy**

  Any data (in the form of policies or accounting) must be readable for authorized roles only. Thus, the transmission channel between them must be secured. The requirement prevents eavesdropping by malicious users in the communication path.

- **(E) Integrity**

  Any data or control in the form of policies must be modified only by authorized roles. This requirement brings the need for authorization schemes in our model. A role might be authorized to perform specific operations. Hence, levels of authorization are desirable.

- **(F) Non-Repudiation**

  Any role must not deny full or partial participation in a communication with other role in the model.

- **(G) Denial of Service (Prevention and Recovery)**

  Denial of Service (DoS) attacks interfere with the normal use or management of the communication resources. Metering and charging applications measure the usage of resources and deals with money. As a result, the Meter system should be considered a possible target of such attacks.

- **(H) Audit**

  Basically, Providers and Customers should be able to audit the Meter systems in order to check for theft.

4.5 Tariff-related Control of the Model

In this section we look at a mechanism to perform a translation from a tariff (high-level) to appropriate Metering/Accounting policies (low-level). This is a work in progress with basis on rule-based languages from active database research [7].

4.5.1 Tariff Translation

A tariff/SLA expresses a set of charging and QoS monitoring policies in a declarative way.

However, there exists a binding mechanism between tariff/SLA (high-level) and Meter configuration (low-level). Such relation may be achieved through metrics specification which includes: (a) type; (b) constraints; (c) implementation scheme; (d) associated event; (e) pattern matching filters.

The language that we have designed is able to express metrics such as (a) traffic-related (e.g. session duration, volume, loss, throughput, delay, jitter); (b) environment-related (e.g. congestion indication); (c) classification-based such as DiffServ DSCodePoint, protocol-related, address and time attributes.

4.5.2 Simple Example

Here we present a simple arbitrary congestion-based tariff with event-condition-action (ECA) rules. It relies on the ECN (Explicit Congestion Notification) proposal [8] as a means for getting congestion feedback from the network.

```
price_byte_unit = 0.4;
ecn_marks: counter;
bytes_sent:  counter in bytes;
on ecn_marks.congestion
   if (price < 1000)
     do {
       price = ecn_price + price_byte_unit*bytes_sent;
     }
```

The above tariff is translated to the following Metering/Accounting policies:

```
on initialization
   do {
     create counter name ecn_marks bound to
        packet_arrival.pkt[NETWORK_ECN_BITS] = 11;
     create counter name bytes_sent bound to
        packet_arrival.pkt.length;
   }
on packet_arrival
   do {
     update bytes_sent;
     update ecn_marks;
     prepare output;
   }
```

In this example, both ecn_marks and bytes_sent variables are counter types. The tariff translator uses this in order to generate the low-level policies.

4.6 Meter System Performance

We have performed initial experiments to assess meter calibration metrics such as Meter Error Rate (MER) and traffic-related metrics such as Volume(V) and Packet Rate (PR). We also have used a tariff based on the IETF DiffServ architecture [9].

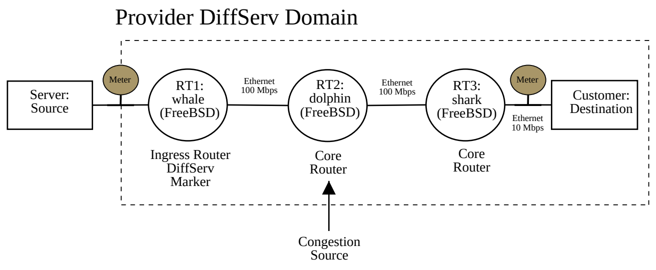

Figure 4.5: DiffServ Testbed

4.6.1 Tariff Description

An arbitrary tariff was delineated with basis on six classes as described in [10]. We have done the mapping from that model to IETF DiffServ classes (Table 4.1). It had considered the traffic characteristics of each class (interactive realtime, non-interactive realtime and non-realtime).

4.6.2 Methodology

We have generated traffic for six distinct application categories shown in Table 4.1. When alternating the meter position throughout the testbed we can assess the Volume (V) and Packet Rate (PR) metrics.

The testbed used in the experiment is shown in Figure 4.5. The server (source) and the customer machine (destination) run Linux. Router RT1 marks ingress packets with DSCodePoint in respect of categories in Table 4.1. A multi field (MF) classifier operates at RT1 mapping applications into the desired DSCodePoint. Three experiment scenarios were used, mainly varying two variables: meter position (source or customer) and network state (congested or not). The scenarios are: (1) Meter at Customer and Network without Congestion; (2) Meter at Customer and Network Congested and (3) Meter at Source.

4.7 Results

Some interesting results might be deduced from the data collected. There is relative discrepancy in the actual metered volume. Although is true to say that, we do not know whether this holds or not regardless of meter positions. Thus, more experiments switching meter places over the Provider network (IETF DiffServ domain in Figure 4.5) need to be be performed. We want to work out an association between the volume metered, generated and charged in congestion periods. At present, we define the volume relation being the following equation:

$$V_{met} = (1 - x).V_{gen} = (y).V_{chg} \tag{4.1}$$

where:
V_{met}: Volume metered, V_{gen}: Volume generated, V_{chg}: Volume charged, x: Meter error rate, y: Volume charged coefficient

It should be pointed out that those volumes are semantically distinct. The parameter x in the equation 4.1 was determined by the experiment data (meter error rate). It has floated around 5% on average according to Fig. 4.6. The possible cause may be linked to the FreeBSD Kernel (meter machine) difficulties for handling all packets seen.

The ambition of assessing DiffServ classification-based metrics has been reached successful. This may prove that we can meter and charge for such metrics.

Application Category	**DSCodePoint**	**Examples**
1: (Gold, Interactive Realtime)	EF=0x46	IP-Telephony (high QoS)
2: (Gold, Non-Interactive Realtime)	AF_{11}=0x10	Audio/Video streaming (high QoS)
3: (Silver, Non-Interactive Realtime)	AF_{12}=0x12	Audio/Video streaming (low Qos)
4: (Gold, Non-Realtime)	AF_{13}=0x14	Web browsing
5: (Silver, Interactive Realtime)	AF_{21}=0x18	IP-Telephony (low QoS)
6: (Silver, Non-Realtime)	AF_{22}=0x20	E-mail

Table 4.1: Classes used in the tariff

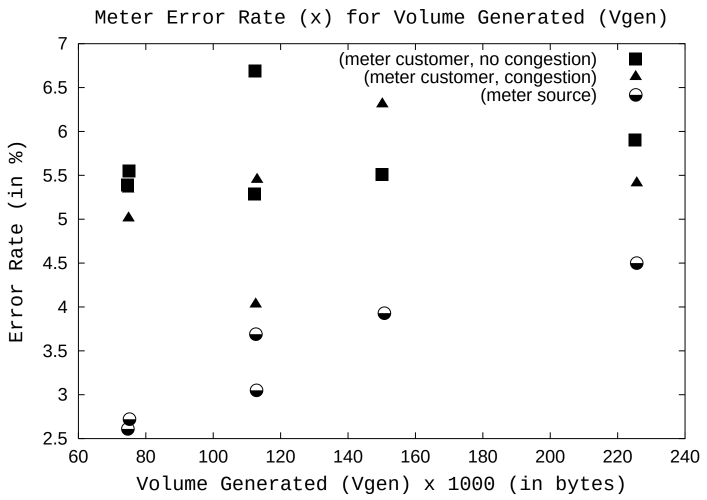

Figure 4.6: Meter Error Rate(x) over Volume Generated (Vgen)

Acknowledgments

We are grateful to Jon Crowcroft, Saleem Bhatti and Tristan Henderson from UCL for comments and discussions. Also, special thanks to Arnaud Jacquet, Kennedy Cheng and Bob Briscoe from BT Labs for their valuable feedback.

References

[1] Franco Travostino, "Towards an Active IP Accounting Infrastructure," in *Third IEEE Conference on Open Architectures and Network Programming - OpenArch 2000*, Mar. 2000.

[2] Michael Fry and Atanu Ghosh, "Application Level Active Networking," *Computer Networks*, vol. 31, no. 7, pp. 655–667, 1999.

[3] Marcelo Pias, Nelson Duarte, and Jon Crowcroft, "Implementation of an Application Level Active Network," URL: http://www.cs.ucl.ac.uk/staff/m.pias/project/alanImpl.

[4] David M. Chess, "Security Issues in Mobile Code Systems," in *Mobile Agents and Security; G. Vigna (ed.)*. 1998, Springer-Verlag.

[5] George C. Necula and Peter Lee, "Safe, Untrusted Agents using Proof-Carrying Code," in *Mobile Agents and Security; G. Vigna (ed.)*. 1998, Springer-Verlag.

[6] Wayne Jansen and Tom Karygiannis, "Mobile Agent Security," Tech. Rep. Special Publication 800-19, National Institute of Standards and Technology (NIST), Aug. 1999.

[7] Norman W. Paton (Ed.), *Active Rules in Database Systems*, Springer-Verlag, New York, 1998.

[8] K. Ramakrishnan and S. Floyd, "RFC 2481 - A Proposal to Add Explicit Congestion Notification (ECN) to IP," *IETF*, Jan. 1999.

[9] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "RFC 2475 - An Architecture for Differentiated Services," *IETF*, Dec. 1998.

[10] M. Alfano, M. Krampell, and M. Smirnov, "End-to-End Quality in IP networks: Can we offer and charge it?," in *XVII World Telecommunications Congress, Birmingham (UK)*, May 2000.

Sphere: A Binding Model and Middleware for Routing Protocols

Vassilis D. Stachtos, Michael E. Kounavis and Andrew T. Campbell
genesis@comet.columbia.edu

The design, deployment and architecting of new routing protocols is time–consuming and costly. Routing protocols are complex systems that are characterized by the way they manage distributed network state in order to formulate forwarding tables. Existing routing protocol implementations are monolithic, bundling together a database with an optimal path calculation algorithm and a network state distribution mechanism. In this paper, we present a binding model and middleware that decomposes routing protocols into fundamental building blocks called Sphere. We identify the role of each component and show how new routing protocols can be dynamically composed from a hierarchy of base classes.

5.1 Introduction

Existing routing protocols are implemented in a monolithic manner and are highly integrated into each vendor's equipment. This practice makes the introduction of new routing protocols or the modification of existing ones difficult. The dynamic introduction of new routing services into the Internet represents a difficult problem. First, routing programming interfaces need to be flexible enough to allow for the introduction of new services. Second, programmers need to understand when and where new services should be introduced. In this paper we argue that routing services can be realized as collections of distributed objects that create associations with each other at run-time. We introduce a binding model that decomposes routing protocols into fundamental building blocks, define programmable objects for routing protocols and use these objects to construct well known (i.e., distance vector, link state, path vector) as well as new routing services.

To formulate our binding model we have studied the design and implementation of existing Internet routing protocols [4-6]. We have found a set of attributes that are common to routing protocols and identified the role of each attribute in the process of formulating forwarding tables. We observe that routing protocols use some type of database. This can be a link state database (e.g., as in the case of OSPF), a distance vector database (e.g., as in the case of RIP), or a path vector database (e.g., as in the case of BGP). In addition, routing protocols use some mechanism for announcing routing information inside the network. This mechanism can be link state flooding such as in the case of OSPF or periodic announcements to neighbor routers as in the case of RIP. Finally, routing protocols use different algorithms and metrics for calculating optimal paths to various destinations. We believe that routing protocol implementations should separate the routing database from the routing information announcement and the optimal path calculation introducing standard interfaces between these components. This technique allows protocol developers to create routing architectures in a modular fashion, and Internet Service Providers (ISPs) to introduce new routing services into their networks more dynamically. Such a separation could provide a foundation for the programmability of routing protocols for the Internet.

Programmable [7-9] and active [10-13] networks represent an emerging area of research. In [7-9] programmable network testbeds and toolkits are described that model the telecommunications hardware using a set of open programmable interfaces. Here there is an emphasis on service creation with quality of service (QOS), and a clear distinction between information transport, network control and network management. Active networks [10-13] allow for the modification of the behavior of network nodes at run-time and offer a higher degree of programmability at the cost of increased complexity of the programming model. A system that supports programmable routing logic for wireless ad hoc networks is discussed in [14]. In [15] a programmable routing protocol scheme is presented that allows extensible link-state routing. Routing protocol extensibility in [15] is restricted to the confines of the link state protocol framework used.

While the community has investigated service creation with QoS and code mobility in programmable and active networks, little work has been done on the programmability of routing protocols. In our work, we have identified similarities between a number of routing protocols and designed a routing Software Development Kit (SDK) and middleware for programming routing protocols called Sphere. We are currently implementing three different routing protocols using Sphere and plan to use the Genesis Kernel [16] to dynamically introduce these routing protocols into our testbed [17]. The Genesis Kernel is a programming system that automates the process of creating deploying and managing network architectures. Sphere is a binding model and middleware programming system designed specifically for the development of new routing protocols, and can be viewed as a kernel plugin or subsystem in the context of the Genesis Kernel – a form of 'routeware'.

The paper is structured as follows. In Section 5.2, we provide an overview of routing protocols that are used in the Internet today. Furthermore we describe the Sphere binding model and the way routing protocols can be decomposed into fundamental building blocks. In section 5.3, we discuss how routing protocols can be programmed using this binding model and building blocks. In Section 5.4, we provide some concluding remarks.

5.2 Sphere

5.2.1 Routing Protocol Overview

Routing protocols found in the literature differ significantly. For example, consider two well-known intra-domain routing protocols: RIP and OSPF. Although these protocols both operate on the same type of networks and their task is similar (i.e., to route packets based on the optimal path between a pair of nodes) each protocol takes a distinct approach to the formulation of forwarding tables. RIP performs the computation of the optimal routes in a distributed fashion between autonomous system (AS) routers using the Bellman-Ford algorithm [17]. OSPF floods information about adjacencies to all routers in the network where each router locally computes the shortest paths by running the Dijkstra's algorithm. BGP, the inter-domain routing protocol, is based on a different design approach, where backbone routers exchange routing information in terms of path vectors [6]. Path vectors are used for loop prevention and policy-based routing and include the complete list of autonomous systems to each destination.

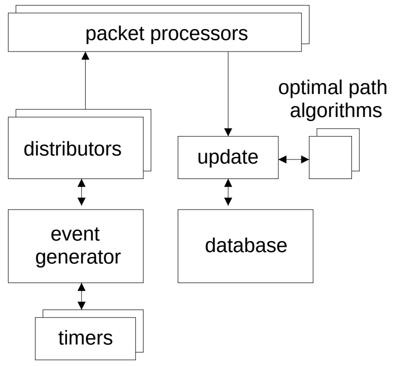

Figure 5.1: Binding Model

5.2.2 Binding Model

A binding model characterizing each routing protocol is illustrated in Figure 5.1. The binding model reflects the structure of routing protocol implementations in a single network node. The components of the Sphere binding model include:

- *a database object*, which is used for maintaining distributed routing state. It can represent a distance vector, link state or path vector database;
- *an update object*, which updates the database when new routing information is received. The manner in which the database is updated is dependent on the routing protocol being programmed;
- *a set of optimal path algorithms*, which operate on the contents of the database to calculate forwarding tables. Each algorithm may use a different metric and operate over a different timescale;
- *an event generator*, which initiates the transmission of routing information, updating of the database or the calculation of optimal paths. The event generator is programmable and can be triggered by timer objects;
- *distributors*, which disseminate routing information in the network. Distribution systems can simply send updates periodically as in the case of RIP or implement complex protocols such as the 'hello' and 'flooding' protocols as in the case of OSPF; and
- *packet processors*, which process routing packets before they are transmitted to the network or after they are received from the network.

By introducing standard interfaces between these routing components we enable code reuse and ease the process of developing new routing protocols. Component interfaces are independent of the specific routing protocols being programmed. By allowing components to create bindings at run-time we enable the dynamic introduction of new routing services into networks.

5.2.3 Routeware: An SDK for Building Routing Protocols

To realize our binding model we have designed a routing SDK called routeware consisting of a hierarchy of base classes, as illustrated in Figure 5.2. Routeware classes can be implemented using object oriented programming languages and environments such as C++, CORBA and Java. We are currently implementing routeware using Java.

Generic Classes

Routeware consists of three groups of classes. At the lowest level, a set of generic classes offers basic functionality to the layers above. A database class supports generic methods for creating new databases and for adding, removing or searching entries. An authentication algorithm class implements a set of authentication algorithms used by routing protocols (e.g., the MD5 algorithm), while a timer management class supports a simple start/stop/reset timer. The network connection class is more complex since it provides the mechanisms used by the protocol at the lowest level for transmitting routing messages to the network. A range of diverse communication mechanisms are supported including TCP and UDP socket management and remote method invocations. Because many routing protocols are multithreaded, a thread management class wraps operating system specific methods for creating and managing threads.

Routing Component Classes

The next layer of abstraction contains routing component classes. Routing component classes represent building blocks that can be used to construct different routing architectures. Routing component classes are either new classes or extend the generic classes described above. A routing database extends the generic database class encapsulating all the route entries available at a network node. The routing database class provides information about network nodes and their associated paths, which can be used for performing route entry lookups. We use a generic data structure to represent a routing database entry. An update class performs the steps necessary for a generic type of routing update. A distributor class retrieves all the route entries that have to be transmitted, sets up the necessary network connections, and finally sends the update information. This is a generic class that can be further extended or used as part of more complex protocol-specific distributor classes.

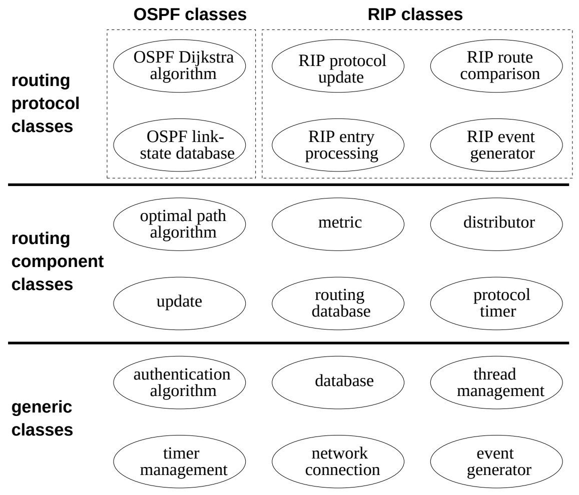

Figure 5.2: Routeware

A metric class defines different types of metrics and a set of fuzzy comparison methods for these metrics. Different types of metrics can be combined to support customized routing policies. A protocol timer class extends the generic timer class to support a range of timer events used by routing algorithms. Timer objects bind to routing protocol specific routines that handle events generated by timers. An optimal path algorithm class provides abstractions used for configuring algorithms that calculate forwarding tables. Programmers can use this class to specify families of protocols associated with optimal path algorithms, types of metrics or database fields. Optimal path algorithm class is an abstract class. Each routing protocol is expected to extend this class in order to define the core of the algorithm.

Routing Protocol Classes

Routing protocol classes use other routeware classes to implement specific routing protocols. Routing protocol classes fully describe the objects that participate in a routing protocol implementation. Examples of these classes, specific to the RIP and OSPF protocols, are illustrated in Figure 5.2. A RIP protocol update class implements routing updates that are specific to RIP. Another class could implement link state updates specific to OSPF. Each protocol is associated with a protocol entry processing class. For example the RIP entry processing class supports methods for processing RIP route entries received from the network. The operations implemented here concern route entry processing only. The reception of packets containing these entries and the extraction of entries is handled by other generic classes described above.

Protocol-specific event generator classes (e.g., the RIP event generator class) interact with timers to generate events when needed. Timer-related constants can be set and the distribution of the interval between two subsequent events specified in case this interval is stochastic. Event generator objects can be configured with information about the actions taken in the case of various events. For example, a RIP event generator can be programmed to call a regular update when the corresponding timer expires. Finally, route comparison classes provide methods for comparing route entries. Route comparison classes make use of metric classes to produce results. Optimal path algorithm classes (e.g., the OSPF Dijkstra algorithm class) make use of metric classes.

The classes described above can be used by many well-known routing protocols. Some routing protocols require additional components. In the case of link-state protocols for example, a set of classes describing flooding mechanisms are needed. Flooding protocol classes extend a generic distributor class (shown in Figure 5.2) using functionality provided by the lower layers. For example, a flooding protocol class can make use of a database class in order to retrieve entries for transmission. Similarly, a flooding protocol class can make use of a network connection class in order to create the necessary end-to-end connections to transmit routing entries.

5.3 Routing Protocol Composition

In what follows, we show how the building blocks discussed in the previous section can be used to construct routing protocols. We begin by examining intra-domain routing protocols focusing on RIP as an example. In Section 5.3.2, we discuss inter-domain routing protocols focusing on BGP.

5.3.1 Intra-domain Routing

The part of a programmable RIP implementation (shown in Figure 5.3) that directly communicates with the network is the network connection object. The network connection object uses UDP sockets to send or receive data to and from other routers. Another variation of RIP could use remote method invocations (e.g., CORBA IIOP, Java RMI). In the case of sockets, a RIP packet processing object is involved which reads routing protocol specific headers from received packets and applies appropriate incoming filters in order to verify the origin and the validity of the received packets.

In the case of remote method invocations, content processing is not applied to headers but on the arguments of remote method invocations. An authentication password is passed to the object that implements authentication algorithms in order to perform validity checks and notify the RIP packet processing object. The algorithms implemented can be simple password or MD5 cryptographic checksum. These algorithms are used by many routing protocols including EIGRP, OSPF and RIPv2.

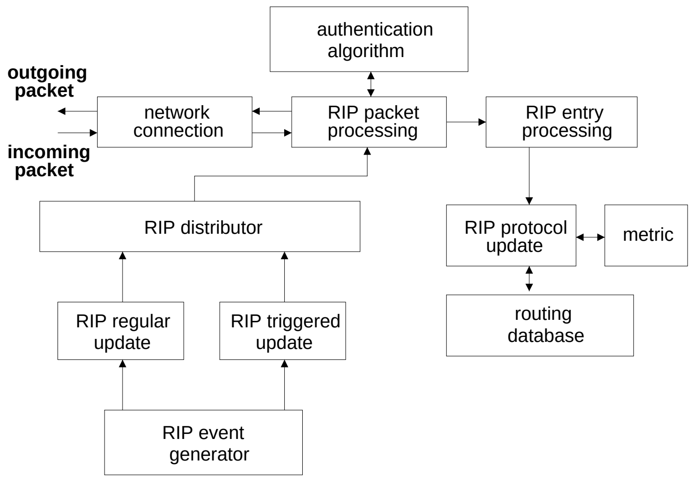

Figure 5.3: Programmable RIP

Once a packet has been examined and verified a route entry is extracted and sent to the RIP entry processing object. The RIP protocol update object is then invoked, which in turn exchanges information with the metric object and the routing database. The routing database replies with the route entry for the same node (if one exists). Following this, the metric object inputs the two entries and produces the comparison result based on a specified set of metrics. If the received entry is better than the existing one the routing database object is used to replace the existing entry. Following this, the RIP event generator initiates a RIP triggered update. Updated entries are finally transmitted to the neighboring routers via the network connection object.

Regular RIP updates are periodically initiated using a distributor object, as shown in Figure 5.3. The distributor object enforces the 'split horizon' [18] principle to the way routing entries are announced in the network. The split horizon principle suggests that announcements concerning destinations should not be sent to routers that are in the shortest paths to these destinations. This feature makes RIP more robust against routing loops and link failures.

Link state routing architectures can be realized using the same binding model and base classes as in the case of RIP. The mechanisms for distributing and processing route entries are different, however. For example, an incoming OSPF packet needs to traverse a packet processing object specific to OSPF. State advertisements need to be extracted and entered in a link-state database. A shortest path algorithm object can apply Dijkstra's algorithm to the contents of the link state database in order to calculate forwarding tables. If there is a need for an update, a flooding protocol can make use of the network connection object to communicate with other routers in the network.

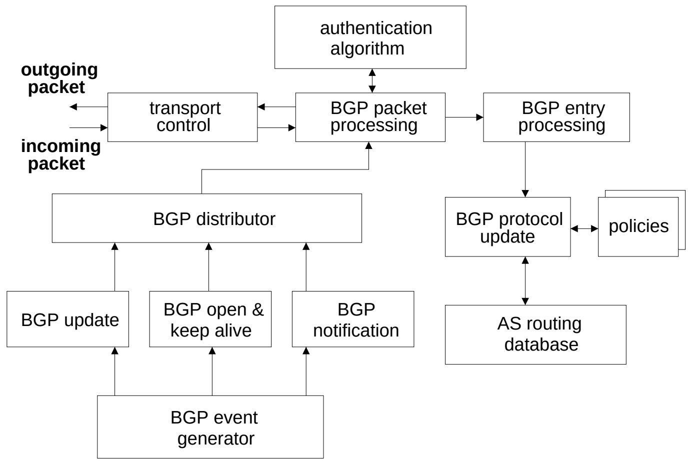

Figure 5.4: Programmable BGP

5.3.2 Inter-domain Routing

Inter-domain routing protocols can be implemented by combining objects from routeware, as shown in Figure 5.4. In the case of BGP, a transport control object extends the network connection class, providing the reliable data transfer mechanisms required by BGP messages. The transport control object receives path entries from the network in the form of BGP update messages. Path entries are processed in a manner similar to the way RIP processes route entries. Each entry is compared with the corresponding entry for the same destination at the local AS routing database. The AS routing database maintains information about destination autonomous systems, their associated paths and their path attributes. The shortest path to a destination is selected by default. Alternatively, the network administrator can program customized policies for selecting the best path. Policies can be implemented as routeware objects that dynamically bind to the BGP protocol as shown in Figure 5.4. A BGP protocol update object compares path entries selecting the best path to an autonomous system.

Other features of the BGP protocol are supported as separate routeware objects. Open, update, keep alive and notification messages are sent by their associated routeware objects, as shown in Figure 5.4. Protocol enhancements can take place with simple software upgrades. For example, one can add support for classless inter-domain routing (CIDR) into BGP by replacing the BGP update and entry processing objects shown in Figure 5.4. These objects can be replaced by other objects that exchange network layer reachability information (NLRI) [6] in addition to the path entries contained in the update messages. Network layer reachability is a feature introduced in the fourth version of BGP for route aggregation.

5.4 Conclusion

We have argued for the construction of routing protocols by extending and combining a set of components. This leads to an open programmable design that can greatly facilitate the design and deployment of new routing architectures. We have proposed a set of components and outline their interaction in order to construct a number of routing protocols. The proposed scheme can be used for both interior and exterior gateway protocols and for different types of routing algorithm. By being able to dynamically program routing protocols we can evaluate and change their structure as needed. Moreover, by doing this on the fly, we can dynamically change the routing policies associated with flows at a router or differentiate their service by loading different modules for each flow, thus applying different routing procedures. We are in the early stages of our work on routeware. Currently we are prototyping the Sphere binding model as part of a routeware plugging to the Genesis Kernel [17].

References

[1] Merit GateD Consortium, `http://www.gated.net`

[2] GNU Zebra – routing software, `http://www.zebra.org`

[3] Cisco IOS Rel.12.1, `http://www.cisco.com`

[4] G. Malkin, ”RIP version 2”, *IETF Network Working Group RFC 1723*, November 1994.

[5] J. Moy, ”OSPF version 2”, *IETF Network Working Group RFC 2328*, April 1998

[6] Y. Rekhter, and T. Li, ”A Border Gateway Protocol 4 (BGP-4)”, *IETF Network Working Group RFC 1771*, March 1995.

[7] Lazar, A.A., ”Programming Telecommunication Networks”, IEEE Network, vol.11, no.5, September/October 1997.

[8] A. T. Campbell, M. E. Kounavis, and R. R.-F. Liao, ”Programmable Mobile Networks”, *Computer Networks*, Vol. 31, No. 7, pg. 741-765, April 1999.

[9] J. van der Merwe et al., ”The Tempest – A Practical Framework for Network Programmability”, *IEEE Network Magazine*, 12(3), May/June 1998.

[10] Tennenhouse, D., and Wetherall, D., ”Towards an Active Network Architecture”, Proceedings *Multimedia Computing and Networking*, San Jose, CA, 1996.

[11] D. Wetherall, J. Guttag, and D. Tennenhouse, ”ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols”, *IEEE OPENARCH’98*, San Francisco, CA, April 1998.

[12] J. Smith et al., ”SwitchWare: Towards a 21st century Network Infrastructure”, `http://www.cis.upenn.edu/~switchware/`

[13] Yemini, Y., and Da Silva, S, ”Towards Programmable Networks”, *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, L’Aquila, Italy, October, 1996.

[14] C. Tschudin, H. Lundgren and H. Gulbrandsen, ”Active Routing for Ad Hoc Networks”, *IEEE Communications Magazine*, April 2000.

[15] C. Partridge, A. Snoeren, T. Strayer, B. Schwartz, M. Condell, I. Castineira, ”FIRE: flexible intra-AS routing environment”, *ACM SIGCOMM*, Stockholm, August 2000

[16] M. E. Kounavis, A. T. Campbell, S. Chou, F. Modoux, J. Vicente and H. Zhuang, ”The Genesis Kernel: A Programming System For Spawning Network Architectures”, *IEEE Journal on Selected Areas in Communications*, to be published, 2001.

[17] The Genesis Project, `http://comet.columbia.edu/genesis/`

[18] C. Huitema, ”Routing in the Internet”, *Prentice Hall*, pp. 529-551, March 1995.

**The Short Paper Session is part of OPENARCH 2001,
the 4th IEEE Conference on Open Architectures and Network Programming,
April 27-28, 2001, Hilton Anchorage Hotel, Anchorage, Alaska. Previous conferences were held in
Tel-Aviv 2000, New York 1999 and San Francisco in 1998.**

**OPENARCH 2001 is sponsored by the IEEE Communications Society
and with support from Intel Corporation and Nortel Networks.**

Editor's address:
Christian F. Tschudin
Dept of Computer Systems, Uppsala University
Box 325, SE-75105 Uppsala, Sweden
`<tschudin@docs.uu.se>`