Netlets: The Future of Networking?[1]

Martin Collier
Broadband Switching & Systems Laboratory
Dublin City University, Dublin 9, Ireland.
collierm@eeng.dcu.ie
tel: 353 1 704 5135
fax. 353 1 704 5508

[1] Presented at the First IEEE Conference on Open Architectures and Network Programming (OPENARCH 98), San Francisco, 3-4 April 1998.

Netlets: The Future of Networking?

Martin Collier
Broadband Switching & Systems Laboratory
Dublin City University, Dublin 9, Ireland.
collierm@eeng.dcu.ie

Introduction

This paper describes research being conducted in the Broadband Switching & Systems Laboratory at Dublin City University into new approaches to networking. It describes our perspective on the crisis in networking, and our work on a new direction for implementing networking protocols in the future.

Let us begin by describing some problems with today's nets, with which this audience will be familiar.

Problem 1:	Network equipment does not support protocol used by application
Solutions:	• Persuade vendors to implement protocol (6 months << $T_{rollout} \leq \infty$) • Tunnel through network to supporting nodes (since these are typically in a lab environment, application may never emerge into the "real world"; also testing scaling issues is difficult) • Revert to legacy protocol (the new protocol probably wouldn't work, anyway) • Reprogram the network (How?)

Problem 2:	Research lab wants to develop and deploy new protocol
Solutions:	• Design and build own network equipment (instructions available at http:\\www.re-invent_the_wheel.com) • Use workstation or other general purpose computer as network equipment (this works, but with hardware unrepresentative of "real world") • Establish research agreement with vendor (6 months persuading tech dept., another 6 months persuading legal dept., 6 months replicating the vendor's toolchain, another 6 months reminding tech dept. of agreement, …) • Develop protocol on open network (What does "open" mean?).

Problem 3:	Every vendor has its own implementation of (say) IQ.239805.1(B) V5.0.
Solutions:	• Who cares? That's what Interop tests are for (this ignores "social" cost – in the sense used by economists – of having the finite population of programmers with network skills independently solving the same problems, whilst other problems are not addressed) • Everyone licenses the Acmetech™ stack (but still has to port it to a unique platform) • Actually, our vendor currently supports V4.2. V5.0 will be available in the third quarter (see Problem 1) • Make every vendor use the same RTOS and controller/hardware interface – then porting software will be easier (this would stifle innovation, and who would regulate it?) • Market should demand vendor support for open network (open how?)

The common solution in each case is to build an open network. How do we build this open network? Firstly, each node in the network (apart from legacy nodes) must support *customer programmability.* This means that the customer (or a third party on his/her behalf) must be able to program the node without special knowledge of proprietary hardware and software features. This requires new approaches to the design of the software for switches and routers which are currently *vendor programmable* and customer-configurable. Secondly, there must be a method for deploying the customer's program in the network.

Programming the net

Customer programmability (or, in general, third-party programmability) may be implemented in a number of ways.

1. Provision of software development facilities "on-board", together with a standard API.

   This would add considerably to the cost of network equipment, but, conceivably, it would be possible to design the switch or router of the future with a plug-in software development module as an optional feature, in the same way that today one can get an optional redundant controller card or PSU. Equipment without such a board could upload run-time code, but could not compile or debug programs.

2. Use of distributed computing approaches, notably CORBA (e.g., xbind [1]).

3. Use of a common logical model of the hardware at a node.

   The Java Virtual Machine is an obvious candidate here. Java is increasingly being promoted for use in embedded systems. It is to be presumed that the level of real-time performance required in networking applications will be supported by Java in the near future, as compiler technology improves, and specialised hardware is rolled out.

It is imperative that the technique adopted uses portable code, since otherwise we are adding complexity where we sought to remove it.

Code Delivery

A second issue concerns the deployment of the code. In particular, who initiates the deployment of code? Possible approaches include:

1. Use legacy methods. This is business as usual as far as network managers are concerned. The only change from existing practice is that third-party software can be installed. This is potentially a considerable boost to innovation in networking, but does not fully exploit the potential of an open network architecture.

2. Use one or more "protocol servers". These are depositories of open code components to support various protocols. A switch receiving a request from an end-user for the use of a protocol which it does not support generates a request to the protocol server, which delivers the required components. Obviously many issues need to be addressed in this approach, such as how to support real-time services, how the population of components is generated and controlled, how the switch determines which components to cache, etc.

3. Use of "push" technology. In a conservative approach, this is just a refinement of legacy methods, to allow network managers to deploy a consistent implementation of a protocol across a multi-vendor network. In a more radical strategy, end-users can push protocol components required to support their application into the network. This raises issues of security and authentication, but would allow applications to be freed from their ties to legacy networking protocols.

4. Switching of objects. The objects would comprise end-user data, state information, and methods implementing the protocol. Capsule switching exemplifies this approach [2]. This is perhaps the boldest vision of an open, active architecture, and the migration path from the current networking paradigms to the "pure objects" paradigm will require careful study.

5. Use of nomadic components (called netlets, by analogy with applets). This is similar to the "protocol server" approach, except that there is no centralised server – the server function is distributed across all nodes of the network. Hence, the single point of failure in the earlier

approach is avoided, although the dynamics involved in locating a required component and delivering it to the requesting node will need careful study.

6. Use of virtual patchboard. The switch or router is not customer programmable in the sense of allowing a customer process to run on board. Instead, remote access is allowed to a wealth of configuration settings using a "virtual patchboard" which is accessed in a standardised way. The patchboard allows data flows which require functionality not natively supported by the switch to be routed through dedicated servers (e.g., farming out signalling requests associated with an unsupported protocol to a dedicated engine elsewhere on the network. This approach is an extension of that used in, for example, GSMP, and, on a longer time scale, by existing network management tools.

Our approach

The Broadband Switching & Systems Laboratory at DCU is investigating the "nomadic components" or netlets paradigm. Our work is focused initially on its use in ATM networks, for the following reasons:

1. This is our core expertise (cf [3]).
2. We have more ATM kit than legacy kit in our campus network.
3. The project dovetails nicely with our SMARTS (Signalling Mechanisms to Accommodate Real-Time Services) project. This latter project is being conducted with the assistance of Sun Microsystems, Inc., who have donated their ATM stack and driver software, and it is hoped to interface the resulting protocol with the Openet package developed by Sun Laboratories [4].
4. Our approach is easier to implement where route pinning is supported. This is already built in with ATM, but QoS-aware routing is still a work-in-progress for IP.

An open network based on this approach must support the following functions:

- **A protocol registration function:**

Nodes at the edge of the network must be able to determine whether the netlets required to support an application are available on the network. It is envisaged that this information can be gathered as part of the link-state approach to routing. A positive result in end-to-end protocol registration would influence the choice of route.

- **Component attraction function**

A mechanism must be put in place such that network nodes requiring components can attract them. Mere flooding would incur too much overhead.

- **Component authentication function:**

Components can only be injected into the network by authorised sources. Agents ("antibodies") will be required to patrol the network to identify and remove alien or corrupted components.

- **Distributed memory**

The network can support only a finite population of components. In the case where a network node with a full complement must load a new component, an existing component or components must be handed off to a neighbour. This process must be managed in such a way as to avoid removal of wanted components.

- **Scavenging**

Agents ("phages") should patrol the network looking for unused components, which can be purged. This provides a means whereby legacy protocols can be retired, and new protocols which are unsuccessful can be withdrawn from service. The criteria for purging a component will need careful study.

Conclusions

This paper has briefly described our approach to open networking. The general architectural details of our approach have been decided, and we are evaluating the use of Java in the implementation. Our architecture should be flexible enough to be extended to object switching, or, alternatively, to the kind of self-organising distributed system postulated by, for example, the Future Systems group at BT Research Laboratories [5]. Our viewpoint is that of the engineer, rather than that of the computer scientist. Comments on the strengths and weaknesses of our approach, as perceived by other researchers in this field, are most welcome.

References

[1] http://comet.ctr.columbia.edu/comet/xbind/overview.html, "xbind overview"
[2] D. Tennenhouse and D. Wetherall, "Towards an active network architecture", *Computer Communications Review*, vol. 26, no. 2, April 1996.
[3] M. Collier, "A three-stage ATM switch with cell-level path allocation", *IEEE Transactions on Communications*, vol. 45, no. 6, pp. 701-709, June 1997.
[4] http://www.sunlabs.com/research/hsn/, "The Openet Architecture".
[5] http://www.labs.bt.com/people/wintercs/alife.htm, "Future Systems Group".