Evolution to the Programmable Internet

Thomas M. Chen, Southern Methodist University

ABSTRACT

This article explores possible transitional steps to add programmability into the Internet: label switching for flexible packet forwarding, open programming interfaces, active packets for dynamic service control, and mobile agents. With a programmable Internet, service providers and users will be able to gain a degree of flexible control over their network services and take advantage of the full potential of the Internet.

INTRODUCTION

The Internet is a programmed network in the sense that IP routers operate by stored program control. In high-speed routers, packet forwarding may be accelerated by special-purpose hardware such as application-specific integrated circuits (ASICs) but software usually controls functions such as routing protocols, Internet Control Messages Protocol (ICMP), and network management. However, routers are generally not *reprogrammable* except by the vendor. Service providers can configure and manage their equipment to provision services, but routers are otherwise closed systems from their viewpoint. Also, from the user perspective, the Internet is a mostly closed system; users may see IP/ICMP packets delivered across the network, but have little direct control over the internal handling of their data.

By opening network equipment to programming (and reprogramming) by the service provider, service providers are freed from the traditional slow process of consensus-based standardization and vendor-dependent implementation. Programmability gives service providers the flexibility to:

- More quickly introduce differentiated or integrated services to support new multimedia applications
- Implement traffic control algorithms to support quality of service (QoS)
- Enhance internetworking functions as needed, for example, translation between media protocols (Internet and existing intelligent-network-based telephony) or different signaling systems (SS7, H.323)
- Move computations into the network for value-added services
- Manage the network more capably than possible with the standard Simple Network Management Protocol (SNMP)

Essentially, Internet capabilities can be expanded and evolved rapidly through software changes without having to upgrade the underlying network infrastructure. This is appealing to service providers, particularly when the future demand for Internet applications is difficult to predict. Various approaches to network programmability are being pursued in fora such as IEEE Project 1520 [1], OpenSig [2], the International Softswitch Consortium [3], and the Multiservices Switching Forum [4].

The full potential of the Internet might be realized by further opening the network to programming control from users. This is an obvious departure from the traditional computer networking paradigm that imposes an (artificial) separation between data communications in the network and data computing in the hosts. If the network is receptive to programming, the Internet may become a virtual extension of the user's computer ("virtual" because of possible conflicts with other users for shared resources). Through this control over network resources, users will be able to customize their packet forwarding service in terms of reserved resources, QoS, and preferential treatment. Furthermore, users will be able to use network resources for distributed processing or storage.

Although the programmable Internet may seem drastically different from today's Internet, we believe that degrees of programmability can be introduced incrementally, and progress is already being made in the development of major elements. Certainly, the necessary technologies exist in the relevant fields of distributed processing, operating systems, object-oriented software, and mobile code. Also, the notion of network programmability has gained broader acceptance since Java has become commonplace in the World Wide Web. Lastly, the potential benefits of programmability demonstrated by the highly successful PC paradigm, in contrast to traditional vertically integrated computer systems, make a compelling case for programmability in the future Internet.

Clearly, it would be infeasible to expect to upgrade the entire Internet infrastructure in one step; instead, a more practical strategy is needed to progressively evolve the Internet by incrementally adding elements for programmability. This article attempts to identify a set of possible transitional steps. We suggest the use of label switching for flexible and programmable packet forwarding. We describe the need for open programming interfaces for faster service creation, being addressed by IEEE Project 1520. We

0163-6804/00/$10.00 © 2000 IEEE

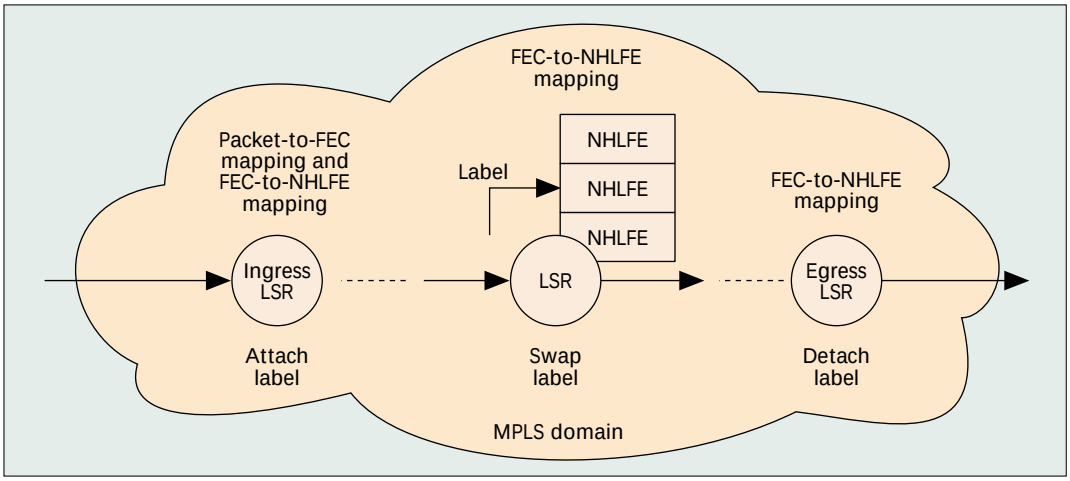

Figure 1. *Packet forwarding in an MPLS domain.*

By opening network equipment to programming (and reprogramming) by the service provider, service providers are freed from the traditional slow process of consensus-based standardization and vendor-dependent implementation.

describe the mechanism of active packets to allow dynamic service control by users. Finally, we discuss programmable mobile agents to carry out delegated user tasks.

Label Switching for Flexible Packet Forwarding

A programmable network has the capability to handle packets flexibly rather than forward them solely on a best-effort basis. In one possible approach, a router could classify incoming packets according to IP source and destination addresses or other header fields, and then execute a set of instructions for packet forwarding. An alternate and more elegant approach is possible taking advantage of the Internet Engineering Task Force's (IETF's) multiprotocol label switching (MPLS) approach [5]. The main difference is that packets are classified upon entrance into an MPLS domain and prefixed with an explicit *label*, a short fixed-length number that is independent of the network layer (e.g., a label does not include any network layer addresses). Within an MPLS domain, the packet is processed and forwarded through the MPLS domain based only on the label using a "label swapping" technique similar to asynchronous transfer mode (ATM) switching. The label is removed by the egress label switching router (LSR) when the packet departs from the MPLS domain. The use of the label eliminates the need for packet classification at each router; classification is done only once at the MPLS domain ingress.

The MPLS approach is a convergence of various implementations of IP switching that use label switching to speed up IP packet forwarding without changes to existing IP routing protocols. Toshiba's Cell Switch Router (CSR) was perhaps the first proposal for an ATM switch that could be controlled by IP protocols rather than ATM signaling protocols. A CSR appears to be an IP router, but can select a flow for cut-through switching at the ATM layer to the next CSR. Ipsilon's proprietary IP Switch was essentially an ATM switch fabric controlled by an external switch controller running IP protocols. Cisco Systems' Tag Switching added a few innovations such as forwarding equivalence classes (FECs), a tag distribution protocol, and stacked tags. IBM's Aggregate Route-based IP Switching (ARIS) was similar to Tag Switching as a control-driven approach, but more specifically designed for ATM switching. MPLS is a consensual implementation-neutral agreement on the base technology for label-switched IP routing.

MPLS-enabled routers, LSRs, can be introduced gradually into the Internet as "islands" of MPLS domains. Separate MPLS domains are interconnected through the existing Internet without having to change equipment outside of MPLS domains. Within an MPLS domain, each LSR examines the label of an incoming packet and uses it as an index into an LSR's forwarding table, as shown in Fig. 1. A label encoding is based strictly on mutual agreement between two neighboring MPLS-enabled routers and has meaning only on the particular link between them. The label can use an existing layer 2 header field — for example, the virtual path/connection identifier (VPI/VCI) field in the ATM cell header — or be inserted between the layer 2 and IP headers as a small *shim* label. A shim label might consist of a 20-bit label value, 3-bit class of service, 1-bit bottom of stack indication, and 8-bit time-to-live (TTL) field to prevent accidental looping.

In the current MPLS specification, forwarding tables consist of next-hop label forwarding entries (NHLFEs) with packet handling information including:

- Outgoing interface (next hop)
- Outgoing label (or push/pop the label stack)
- Data link encapsulation (optional)
- Information about resources (optional)
- Packet handling policies (optional)

More generally, an NHLFE can be viewed as an arbitrary set of instructions detailing how a packet should be treated. An incoming packet with label L causes the instructions in the NHLFE corresponding to index L to be executed on the packet, as shown in Fig. 2. In this view, an LSR can be programmed with any packet handling instructions, and the packet labels serve as a simple index to programming instructions.

The granularity of control depends on the classification of packets at the ingress into an MPLS domain into subsets called FECs. Packets

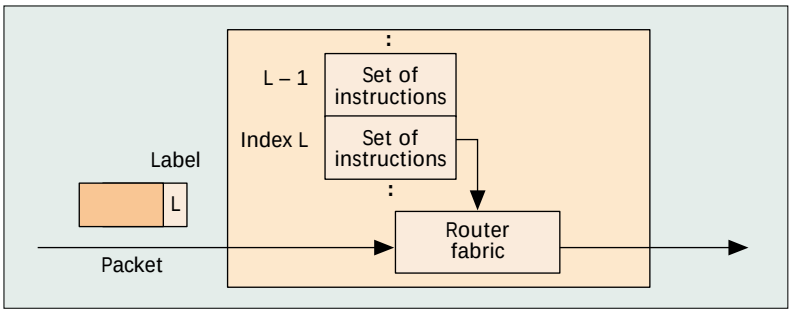

Figure 2. *A programmable router using packet labels as index to instructions.*

belonging to the same FEC are handled in the same manner as by an LSR, but the mapping of packets into FECs can be quite general. The packet-to-FEC mapping can be intentionally coarse (e.g., all packets with the same destination address) or fine (e.g., all packets belonging to a particular application between two hosts), depending on the desired granularity of control.

Although MPLS was developed for high-speed IP routing in the Internet backbone, it can also be a way to gradually introduce islands of programmable switches, but programmability (and its benefits) will be limited to the boundaries of each domain and a granularity of control down to FECs. Also, programmability is limited to packet forwarding and not control; for compatibility with IP routers, MPLS maintains existing control architectures (Internet routing protocols, etc.). For a more ambitious level of network programmability, the control architecture can be opened with standardized programming interfaces.

Programmable Interfaces for Open Service Creation

Open programming interfaces define the boundary between the hardware platform and software control to enable an open service creation environment where services may be able to evolve faster than the underlying network. Agreements on programming interfaces allow third parties to develop application software and service providers to program equipment from different vendors. The OpenSig community has been attempting to standardize "open programmable networks" which has led to the IEEE Project 1520 effort to define and standardize software abstractions of network resources and application programming interfaces (APIs) for manipulation of these resources [1]. The primary concern is the control plane and standardized programming interfaces for *open signaling*. Open signaling is based on the belief that standardized signaling protocols are a crucial element for new multimedia services, but are now too complex and general-purpose. A programmable network with well-defined APIs will open the network to new signaling protocols without depending on consensus-driven standardization of each signaling protocol. Furthermore, without the need to settle on a single general-purpose signaling protocol, different signaling protocols can be optimized for specific applications.

The P1520 reference model for IP routers is shown in Fig. 3. It follows a layered view of a programmable network that is similar to the abstraction of an operating system in a personal computer. An operating system provides programming interfaces for applications to make use of the physical resources of the computer. With an analogous software layer over a programmable network, different control architectures including signaling protocols can operate on the same network.

The lowest layer consists of physical resources for forwarding IP datagrams such as routing tables, buffers, packet classifiers, packet schedulers, and switch fabrics. They may be directly accessed through the so-called connection control and management (CMM) interface. The CMM interface is not intended to be used for programming, but mainly to provide access to management protocols to collect state information. An example of access at this level is Ipsilon's General Switch Management Protocol (GSMP) [6]. GSMP is a master/slave protocol that allows an ATM switch to be controlled by an external computer to establish and release virtual connections, add and delete leaves in point-to-multipoint connections, manage ports, and collect status information and alarms. The ATM switch and controller exchange GSMP messages through an ATM virtual connection identified at initialization. Variable-length GSMP messages consist of five types:

- *Connection management* messages to establish and release virtual connections
- *Port management* messages to activate or deactivate specific switch ports
- *Configuration* messages to discover the capabilities of the switch
- *Statistics* messages to request information on VC or port activity
- *Event* messages to report alarms

Extensions of GSMP are being considered by Project 1520 and an IETF working group on GSMP as an interface for a router to control an ATM switch fabric.

Protocols will work with resources through the higher virtual device layer where the physical resources are represented as abstract objects under the control of middleware. Objects in the virtual device layer are accessed by the higher layer through the lower (L) interface (the main objective of the IEEE P1520 project and key to IP router programmability).

The next higher layer consists of algorithms for routing, packet scheduling, packet discarding, and resource reservation. Different algorithms can be developed for differentiated or integrated services through appropriate manipulation and control of router resources. Resource Reservation Protocol (RSVP) or other application-specialized resource reservation (signaling) protocols can be developed. This layer provides network services to applications at the highest layer through the upper (U) interface.

Active Packets for Dynamic Services

An open programmable network may benefit users through faster introduction of new services, but users still depend on service providers. The approach of active networks assumes a pro-

grammable network infrastructure, and furthermore that the network is receptive to receiving and executing programs from users in the form of *active packets* [7]. Unlike data packets, active packets contain program fragments that are executed at each node. Although computations in the network are possible, the main purpose of the network is expected to be data transport. Active networks are not primarily intended to be a means of distributed processing. Hence, active packets are useful primarily for customizing the user's network service. The benefit for users is a new ability to change their services dynamically and optimize network services for specific applications (e.g., sophisticated packet discarding or scheduling). By essentially "programming" the network, the user has a fine-grained level of control over his/her service, perhaps down to the level of individual packets.

Two distinct approaches are possible for code distribution. In the so-called *discrete* approach, active packets are essentially sent out-of-band. This approach could be evolutionary from today's Internet because data packets are unchanged in format. It may be most appropriate when service changes do not need to be very frequent. In the *integrated* approach, all packets are executable content (i.e., active packets which may or may not include data). This approach enables each packet to determine its specific handling.

In any case, active packets do not necessarily carry the entire program code, which could be costly in bandwidth and delay. Alternatively, it is quite possible for active packets to contain only identifiers or references to invoke preinstalled programs in the network, as is done in SwitchWare [8]. Furthermore, preinstalled programs do not need to be stored entirely at each router (which could be burdensome). Some programs could be downloadable as needed from distributed servers through the Internet, as proposed in Delegated Agents for Network Management (DAN) [9] or the Softswitch Consortium [3]. The necessary code could even be propagated from node to node as needed, as in ANTS [10].

Active packets are evaluated within a safe execution environment analogous to a UNIX shell program. The code could effect packet forwarding procedures or modify the nontransient state of a router. The execution environment provides an interface for requesting and customizing services to be translated into tasks for the node's operating system. For deployment of active networking in the Internet, in addition to a programmable network infrastructure, there need to be common agreements on active packets and execution environments.

MOBILE AGENTS FOR DELEGATED TASKS

Active packets have an obvious similarity to the better-known mobile agents, and serve a complementary purpose in some ways. Although there is no universal definition of a software agent, generally agents are regarded as software programs designed to carry out a specific function or task on behalf of a user with one or more of certain characteristics: continuous operation,

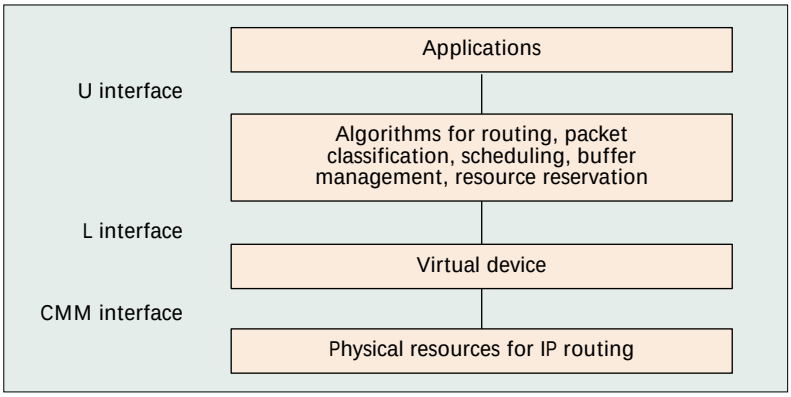

Figure 3. *The IEEE Project 1520 reference model for IP routers [1].*

autonomy, responsiveness to their environment, intelligence (learning), mobility, and communication with other agents [11]. Both active packets and mobile agents view the network as a single distributed programmable environment, but they are different in orientation and operation. Active packets are concerned mostly with customizing the packet handling service related to a user's connection (although not limited in this regard). They work within the network on the network itself, by a process of code movement and remote execution. In contrast, mobile agents may typically have more intelligence and self-directed autonomy to carry out their function, which may be more oriented toward mobile computation (i.e., hosts rather than the network infrastructure). Mobile agents involve not only movement of code, but also suspension of their current execution state (including data or context) and transparent resumption at another location. They require a more sophisticated supporting framework, particularly if multiple agents are allowed to socialize to work together.

In the Internet environment, where an enormous amount of information is distributed throughout a vast network, the mobile agent paradigm is extremely well suited to searching, filtering, and collecting information. By autonomously carrying out delegated tasks, agents can be valuable for empowering users who might otherwise be frustrated by the difficulty of navigating through an overload of information. Among other applications, agents are clearly demonstrating usefulness in electronic commerce, where they can carry out the tedious (and sometimes complex) processes of searching and negotiations through the Internet on behalf of sellers and buyers.

Mobile agents have been very appealing as an approach to the problem of network management. In the current SNMP paradigm, centralized network managers collect status information from the network by repeated polling. This process may result in inefficient communications, long delays to find specific management information, and data overload at the network management system. Mobile agents, an example of the "management by delegation" approach, can overcome these limitations by moving data processing and filtering closer to nodes where the management data resides [12]. Because of the nature of network management, this is an unusu-

The benefits of programmability have been clearly demonstrated in the highly successful PC paradigm, and we believe it is inevitable that new software technologies will transform the traditional thinking about computer networks.

al application of mobile agents to the Internet itself. For network management, mobile agents are an approach conceptually similar to but different from active packets.

Most mobile agent systems use the Internet only as a medium for mobility between hosts. Several mobile agent systems are in use today, such as Telescript, Concordia, Odyssey, Voyager, and Aglets. Although not universal, Java is the prevalent language choice (e.g., Concordia, Odyssey, Voyager, Aglets) due to several reasons: dynamic class loading, multithreading support, object serialization, networking support, and ubiquity of the Java virtual machine.

Unfortunately, most mobile agent systems are currently unable to interoperate due to differences in languages, operating systems, and architecture design. Mobile Agent System Interoperability Facility (MASIF) is a collection of interfaces to enable interoperability between mobile agent system implementations [13]. It attempts to standardize agent management, agent transfer between different systems, agent names for identification, and agent system type and location syntax (for agents to locate each other). It does not specify interoperability between agent systems in different languages.

MASIF relies heavily on the Common Object Request Broker Architecture (CORBA), which allows applications to communicate with each other regardless of location or origin. In 1991, CORBA 1.1 defined the Interface Definition Language (IDL) and API that enables client/server object interaction within a specific implementation of an object request broker (ORB). CORBA 2.0 later specified interoperability between ORBs from different vendors. The ORB is middleware that establishes the client-server relationships between different objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across the network. Users can access information transparently without having to know what software or hardware platform it resides on or where it is located. The ORB intercepts the call and finds an object that can implement the request, pass parameters to it, invoke its method, and return the results. The client does not need to be aware of the object's location, programming language, operating system, or any other system aspects not integral to an object's interface. Client and server objects written in a number of programming languages and run on a number of different platforms can still communicate and interact with each other. With an ORB, the protocol is defined through the application interfaces via a single implementation-language-independent specification, the IDL.

Finally, if multiple agents need to coordinate their tasks (e.g., to implement end-to-end solutions), a common agent communication language such as knowledge query and manipulation language (KQML) is required. KQML is fashioned after human speech interactions, where intent is important as well as message content. The language consists of three layers. The content layer has the actual content in the sender's language. The communications layer handles message features such as sender and receiver identities. The message layer encodes the message, including content and intent, using a set of predefined message types.

CONCLUSIONS

The benefits of programmability have been clearly demonstrated in the highly successful PC paradigm, and we believe it is inevitable that new software technologies will transform the traditional thinking about computer networks. The important question is not so much whether programmability will move into the Internet but how to evolve systematically toward a programmable network infrastructure.

Increasing levels of programmability will allow service providers more flexibility to differentiate and compete, and ultimately will give users more direct control. We have not addressed security here, but security risks clearly become proportionately more serious as control over the Internet becomes more flexible and distributed. Security is a possibly major obstacle to programmability in the network, but if security and safety can keep pace with network technology, users may see the full potential of the Internet realized.

REFERENCES

[1] J. Biswas *et al.*, "The IEEE P1520 Standards Initiative for Programmable Network Interfaces," *IEEE Commun. Mag.*, vol. 36, Oct. 1998, pp. 64–70.
[2] OpenSig, http://comet.ctr.columbia.edu/opensig
[3] Int'l. Softswitch Consortium, http://www.softswitch.org
[4] Multiservices Switching Forum, http://www.msforum.org
[5] B. Davie, P. Doolan, and Y. Rekhter, *Switching in IP Networks*, Morgan Kaufmann, 1998.
[6] P. Newman *et al.*, "Ipsilon's General Switch Management Protocol Specification Version 1.1," Internet RFC 1987, Aug. 1996.
[7] D. Tennenhouse *et al.*, "A Survey of Active Network Research," *IEEE Commun. Mag.*, vol. 35, Jan. 1997, pp. 80–86.
[8] D. Alexander *et al.*, "The SwitchWare Active Network Architecture," *IEEE Network*, vol. 12, May/June 1998, pp. 29–36.
[9] D. Decasper and B. Plattner, "DAN: Distributed Code Caching for Active Networks," *IEEE INFOCOM '98*, San Francisco, Mar. 29–Apr. 2, 1998, pp. 609–16.
[10] D. Wetherall *et al.*, "Introducing New Internet Services: Why and How," *IEEE Network*, vol. 12, May/June 1998, pp. 12–19.
[11] J. Bradshaw, Ed., *Software Agents*, AAAI Press, Menlo Park, CA, 1997.
[12] G. Goldszmidt and Y. Yemini, "Delegated Agents for Network Management," *IEEE Commun. Mag.*, vol. 36, Mar. 1998, pp. 66–70.
[13] D. Milojicic *et al.*, "MASIF: the OMG Mobile Agent System Interoperability Facility," *2nd Int'l. Wksp. Mobile Agents*, Sept. 1998, pp. 50–67.

BIOGRAPHY

THOMAS M. CHEN [SM] (tchen@seas.smu.edu) is an associate professor in the Department of Electrical Engineering at Southern Methodist University, Dallas, Texas. He received B.S. and M.S. degrees in electrical engineering from MIT, and a Ph.D. degree in electrical engineering from the University of California, Berkeley. From 1989 to 1997 he worked on ATM research at GTE Laboratories, Inc., Waltham, Massachusetts. He is a senior technical editor for *IEEE Network*, a technical editor for *IEEE Communications Magazine*, (past) founding editor of *IEEE Communications Surveys*, and editor of *ComSoc e-News*. He received the IEEE Communication Society's Fred W. Ellersick best paper award in 1996. He is co-author of the monograph *ATM Switching Systems* (Artech House, 1995).