The ACTIVE IP Option

David J. Wetherall and David L. Tennenhouse*

Telemedia Networks and Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology

Abstract

In this paper, we discuss our work on an active network architecture in which passive packets are replaced with active capsules — encapsulated program fragments that are executed at each switch they traverse. This approach allows application-specific processing to be injected into the network. The accessibility of computation and storage "within" the network provides a substrate that can be tailored to build global applications, including those that invoke customized multicast and merge processing.

We describe an extension to the IP options mechanism that supports the embedding of program fragments in datagrams and the evaluation of these fragments as they traverse the Internet. The ACTIVE *option provides a generic approach to the extension of the IP network service.*

1 Active Networks

Traditional data networks passively transport bits from one end system to another. Ideally, the user data is transferred opaquely, with the role of computation within such networks being extremely limited, e.g., header processing in packet-switched networks and signaling in connection-oriented networks.

Active Networks [TW96] break with tradition by allowing the network to perform customized computations on the user data. For example, a user of an active network could send a customized video transcoding program to a node within the network (e.g., a router) and request that the node execute that program when processing their packets.

These networks are "active" in two ways:

- switches within the network perform computations on the user data flowing through them;
- individuals can inject programs into the network, thereby tailoring the node processing.

Interoperability

Traditional packet networks achieve interoperability by standardizing the syntax and semantics of packets. Internet routers all support the agreed IP specifications; although router implementations may differ, they implement "equivalent" programs. In contrast, active nodes can execute many different programs.

*{djw,dlt}@lcs.mit.edu. http://www.tns.lcs.mit.edu/

Interoperability is achieved at a higher level of abstraction — instead of standardizing the computation performed on every packet, we standardize the computational model. This provides a platform for user-driven customization, allowing new services to be deployed faster than can be sustained through industry-wide standardization.

From Packets To Programs

We are replacing the passive packets of present day architectures with active "capsules" - miniature programs that are executed at each router they traverse. User data can be embedded within these mini-programs in much the way a page's contents are embedded within a fragment of PostScript code.

Capsules use the built-in constructs of a programming language to perform packet processing. They may also invoke pre-defined primitives that interact with the local node environment; and they may extend and specialize the methods of network-based objects. This allows for both efficiency in the case of the most popular programs, and extensibility in the case of many application-specific ones. Retrospectively, we may come to think of today's routers as implementing a restricted subset of active nodes, i.e., the IP protocol version field may be thought of as a single instruction "program" that invokes the pre-defined `routeIPv4` primitive.

Programming is also facilitated by allowing capsules to leave information behind in a node. Thus, a connection may be opened by having a capsule leave a small amount of associated state at each node along the path it traverses. Subsequent packets can include code whose execution is dependent on this state. Connections in active networks can be more powerful than those of present day systems because the state left behind may be in the form of programs. Flows [Cla88], which are somewhat "softer" than connections, could be realized with a similar scheme.

Advantages of Capsules

In principal, there are three advantages to basing the network architecture on the exchange of active capsules, rather than passive packets:

- exchanging code provides a basis for adaptive protocols, enabling richer interactions than the exchange of fixed data formats;
- capsules provide a means of implementing application-specific functions at strategic nodes;
- the programming abstraction provides a mechanism for infrastructure innovation, allowing new services to be deployed at a faster pace than can be sustained by standardization.

Section 2 provides some background to motivate Active Networks and further details of our approach can be found in [TW96]. In section 3, we describe an implementation of the ACTIVE IP option, a mechanism we are using to explore the issues associated with programming an active network.

2 Motivation

Our work is motivated by both technology "push" and user "pull". The "push" is the emergence of *active technologies*, which provide mechanisms for injecting program fragments into shared resources. Today, active technologies are applied within individual end systems and above the end-to-end network layer; for example, to allow Web servers and clients to exchange program fragments. Our innovation is to leverage and extend these technologies for use within the network.

Although these technologies have been under development for some time, only recently have we begun to see work that addresses the issues of safety, mobility, and efficiency in concert. Java [Gos95] uses an intermediate code representation that is portable and can be statically verified for safety by its consumer. Software-based fault isolation, which can enforce modularity efficiently, is used as the basis of the Omniware system [A+96]. Finally, SPIN uses a language-based approach to achieve safety without sacrificing performance.

The "pull" comes from the ad hoc collection of firewalls, Web proxies, multicast routers, mobile proxies, video gateways, etc. that perform user-driven computation at nodes "within" the network. Despite architectural injunctions against such practices, these nodes are flourishing, suggesting user and management demand for their services. Examples include application-neutral TCP snooping at wireless base stations [B+95a]; "nomadic routers" that are interposed between an end system and the network [Kle95]; and application-specific service gateways that perform file caching and the transcoding of images [A+95].

Our goal is to replace these ad hoc approaches with a generic capability that allows users to program their networks. Active networks address the mismatch between the rate at which user requirements can change, i.e., overnight, and the pace at which physical assets can be deployed. The aim is to accommodate the increasing diversity in networking (e.g. IPv6, mobility, multicasting, wireless media, etc.) by easing the process of innovation. This mirrors trends in extensible operating systems research, such as the Exokernel and SPIN [E+95, B+95b], which facilitate application-specific customization of kernel functionality.

3 The ACTIVE IP Option

Our ultimate objective is to develop a new approach to networking based on active technologies. However, much can be learned by "retrofitting" active capabilities to the existing Internet. To this end, we are extending the options mechanism of the IP network layer to carry program fragments.

We chose IP options as a point of departure because they already serve to aggregate processing that is unrelated to the common case but has, over time, proven useful. Today, IP options are used to specify unusual datagram processing, typically for network monitoring or measurement, e.g., to control the route of a datagram from its source, record the route taken, and gather timing information. They have been previously used in attempts to accommodate diversity [B+94]. Since the options mechanism is being revamped to be both more general and more efficient as part of IPv6 [DH95], experimentation with it is timely.

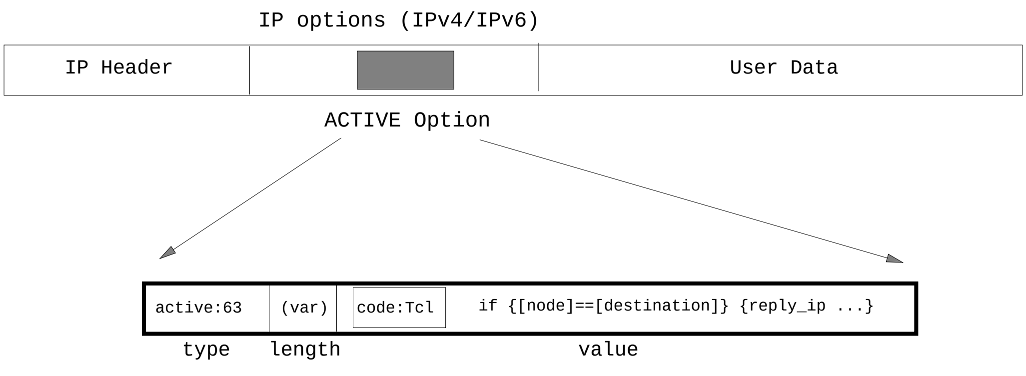

Figure 1: Format of the ACTIVE IP Option Field

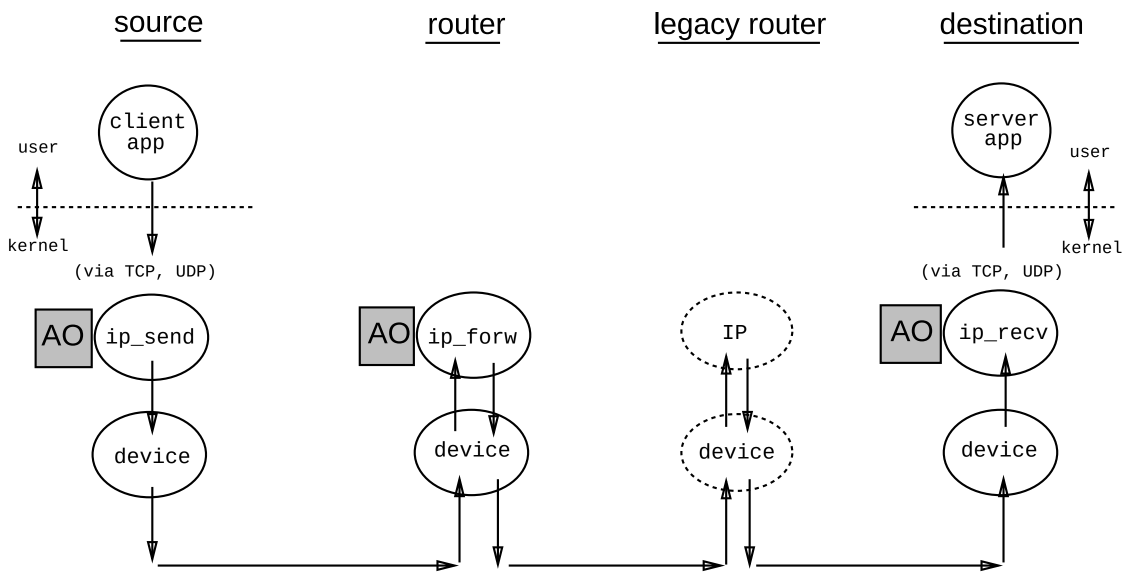

Figure 2: Path and processing of an ACTIVE IP Option Packet

ACTIVE **Option Fields and Processing**

The ACTIVE IP option field (Figure 1) provides a mechanism for embedding a program fragment in an IP datagram. These fragments are then executed by active routers along the path taken by the datagram. Conceptually, the ACTIVE option transforms the IP options from a fixed set of standardized routines to an extensible collection integrated by a programming language.

Our design addresses three high-level issues: how program fragments are embedded in packets; when they are processed; and what processing environment they may expect when they are evaluated.

Two specific option codes which adhere to the generic type-length-value format of IP options [Pos81, DH95] are defined. The first is used to carry program fragments, which may be encoded in a variety of languages. An example fragment is shown in Figure 1. We expect a spectrum of encodings, from source-level through intermediate code to binary representations (e.g. Tcl, Java, and sandboxed portable object code) depending on the required performance versus ease of extensibility. The second option is used to query an active router for the languages it supports. For backwards compatibility, we rely on the RFC 1122 [Bra89] requirement that Internet hosts must silently ignore options they do not recognize.

The ACTIVE option processing model is shown in Figure 2. The processing engine, shown as the shaded rectangle labeled `AO`, sits adjacent to the IP layer, and is invoked from within it as the packet passes through that layer. That is, ACTIVE options do not form a separate protocol layer, and so their processing need not follow a peer-to-peer pattern. Processing occurs once at the source, at each active router that the packet traverses, and at the destination.

ACTIVE options have a great deal of latitude to perform routing, copying, and merge functions. The processing environment allows ambient network conditions to be examined, the current datagram to be dispatched, and additional datagrams to be constructed and launched. The header and payload of the datagram may be read and altered, as may the ACTIVE option fields themselves[1]. The state of the node may also be modified.

[1] Our present implementation does not support rewriting the header, however.

Packet Manipulation	Control Operations	Environment Access
`data` (for access)	`send_ip`	`node` (IP address)
`replace_data`	`send_udp`	`time` (current)
`append_data`	`reply_ip`	`host`
`source` (IP address)	`reply_udp`	`domain`
`destination` (IP address)	`discard`	`mtu`
`length` (of data)	`eval [data 0]`	
`protocol`		
`ttl`		

Table 1: Programming environment during Tcl ACTIVE IP Option processing.

Implementation

We have implemented an ACTIVE option processor within the Linux kernel (2.0.0; Debian) and tested it across a group of PCs (pentium 133MHz or better) connected by 10 and 100 Mbps Ethernet. Our first language encoding is Tcl, which is processed by a stripped-down Tcl interpreter embedded within the kernel. It provides a restricted environment conceptually similar to that of Safe-Tcl [Bor94][2]. User-level code may attach ACTIVE options to outgoing IP packets by using the `setsockopt` system call. This makes them available across all sockets that support the Internet family of protocols, e.g. TCP, UDP, raw IP.

Primitives built into the interpreter and written in C allow access to the current datagram and the other interface functions. Tcl itself is used as a scripting language to tie the primitives together. The primitives are listed in Table 1, and already support most of the core capabilities of generic ACTIVE options, though we intend to flesh out the implementation somewhat.

Several primitives allow the packet itself to be manipulated. One group, e.g. `source` and `ttl`, allow access to individual header fields. Other commands allow portions of the user data to be accessed and updated, either by replacing or adding to the packet; an offset is used to step over any higher layer protocol headers.

Another set of primitives allow new packets to be created. The `reply_ip` and `reply_udp` primitives assume that the current packet is an IP or UDP packet, respectively, and use its addresses to generate a reply, using the current node as source address and including user-specified data. `send_ip` and `send_udp` function similarly, but allow the destination parameters to be given explicitly rather than inferred. Finally, and perhaps most importantly, the option may cause the payload to be evaluated by the interpreter, thus allowing long programs and their state to be embedded in packets.

A third class of primitives allow packets to interact with the local environment. Primitives exist to determine the attributes of the current node (e.g. `node` for IP address) and its network links (e.g. `mtu` for frame size), as well as the current time. It should be possible to expand this set to include other information available through the `/proc` filesystem.

ACTIVE options are executed in a single shared interpreter, and are serially run to completion. Scripting errors cause the packet to be aborted and may be logged locally, but do not trigger additional messages. The current implementation was designed for proof-of-concept, and so is limited in several ways: it is tied to IPv4 with consequent restrictions on the length of options, is written for functionality rather than performance, and at present has no resource management or accounting, e.g. memory is allocated from the shared kernel pool, and execution time is not limited.

[2] Our implementation is based on Tcl7.4. Safe interpreters have since been included with Tcl7.5

```
record route:
  append_data 0 [node]


ping:
  if {[node] == [destination]} then {
    reply_ip [data 0]
    discard
  }


mtu:
  append_data 0 [mtu]
  if {[node] == [destination]} then {
    reply_ip [data 0]
    discard
  }
```

```
trace:
  reply_ip "[node] (ttl=[ttl])"
  if {[node] == [destination]} then {
    discard
  }


probe:
  replace_data 0 "[node] [host] [mtu]"
  reply_ip "[data 0] (ttl=[ttl])"
  if {[node] == [destination]} then {
    discard
  }
```

Figure 3: ACTIVE IP Option program examples, expressed with the Tcl encoding.

Demonstrations

We have implemented several demonstration programs to help us explore this new functionality. The `record route` and `ping` program fragments in Figure 3, for example, correspond to the record route IP option and "ping" ICMP echo request/reply functions. In these and other fragments, the offset of 0 indicates that we are working with an application that sits directly atop IP; an offset of 8 would step over UDP headers[3].

It is relatively easy to construct network probing tools. The `mtu` script, for example, accumulates the link transmission unit (i.e. MTU) sizes along a path and then informs the source. It may be used as a component in path MTU discovery, which is known to have a measurable performance impact [KM87]. The script improves on current MTU discovery techniques by eliminating the need for a search based on ICMP error messages. It determines the MTU directly, and may be especially useful in IPv6, where fragmentation is the responsibility of the end node. The `trace` script provides alternative *traceroute* functionality, tracing the nodes visited by the packet. It takes advantage of the ACTIVE option functionality by using a single probe packet that both generates replies and continues to be forwarded, simplifying the process and shortening its latency. The `probe` fragment demonstrates a script that combines the previous two functions. Many other scripts are possible, and may be useful, for example, as a basis for gathering mobile network traces [N+96].

4 Related Work

We believe our ACTIVE option is the first use of computational objects within the IP network layer. There is much work on mobile code systems, e.g. Java-based Web clients and servers, but such systems operate on an end-to-end basis, passing code across the network untouched.

Under the banner of programmable networks, we are aware of SwitchWare, NetScript and an effort at Georgia Tech [SFG+96, Yd96, BCZ96]. SwitchWare is based on a programmable ATM switch with user-selectable network services. Netscript is positioned as an agent-based programming language for managing networked systems remotely. The work at Georgia Tech focuses on network congestion, selecting preset routines on a per packet basis (and also via information in the IP option field). All of these efforts separate programming the network from using it, whereas our capsule approach is completely in-band, carrying mobile code along with the data on which it operates.

[3] Also, because of option length constraints, the fragments are carried in packets in a more concise form, both by using abbreviations for command names (e.g, `appd` for `append_data`) and abstracting common idioms into Tcl procedures (e.g. `proc here {} {expr {[node]==[destination]}}`).

Other work on extensible operating systems and application-specific networking highlights the increasing safety and efficiency with which user-supplied protocol processing can be implemented. The exokernel [E+95] provides applications with very low-level access to the machine. It has been used to support application-specific safe message handlers (ASHs) [WEK96], which demonstrate the use of user-supplied code and on-the-fly compilation within the context of device drivers and packet filters. Alternatively, SPIN [B+95b] is pursuing a language-based approach to safe kernel extensibility, and Scout [MP96] is providing automatic mechanisms for high-performance network processing based on the "path" abstraction.

5 Conclusions

Our Tcl-based implementation of the ACTIVE IP option has demonstrated a generic facility for programming the network. The ability to tag a packet with a program fragment supports a spectrum of behaviors. In simple usage, the program performs operations on the packet contents, i.e., the separation of datagram header and payload is maintained. In advanced applications, the option program can `eval` the payload, thereby realizing the Active Network paradigm. In the next stage of our research, we intend to implement a Java-based capsule encoding, introduce active storage [TGSK96], and experiment with safety and efficiency mechanisms such as on-the-fly compilation.

We have used the ACTIVE option to construct a set of network probing/discovery functions. It is clear that by using a scripting language to combine primitives, much of the functionality of the existing IP option suite can be subsumed. More ambitious experiments, e.g. those that mimic the application-level video gateway [A+95] and TCP snooping [B+95a], are also possible. ACTIVE options provide applications with a means to attach policy decisions to their data directly, rather than attempting to locate processing at pre-determined positions in the network. For example, with compression routines and a test for available bandwidth, we could script an option that compresses the packet data only if a low bandwidth link is to be traversed.

Active networks involve the synthesis and extension of programming language, operating systems and networking expertise. We realize that suggestions for object-oriented approaches to networking surface every five to ten years. Nonetheless, we are convinced that recent improvements in the safety and efficiency of active technologies, and the demand created by lead applications, present new research opportunities. We also anticipate changes to the organization of end-system software — in place of protocol "stacks", applications may use protocol "components" that can be specialized and composed to perform application-specific functions [CT90].

Acknowledgments

We thank Frans Kaashoek for his helpful advice and comments, and Henry Houh for proofreading this manuscript. Sun Microsystems Inc. provided seed funding for this project.

References

[A+95] E. Amir et al. An Application-level Video Gateway. In *Multimedia '95*, pages 511–522, San Francisco, CA, November 1995. ACM.

[A+96] A. Adl-Tabatabai et al. Efficient and Language-Independent Mobile Code. In *Proc. ACM SIGPLAN Symp. on Programming Language Design and Implementation*, May 1996.

[B+94] T. Blackwell et al. Secure Short-Cut Routing in Mobile IP. In *1994 Summer USENIX*, pages 305–316, Boston, MA, June 1994.

[B$^+$95a] H. Balakrishnan et al. Improving TCP/IP Performance over Wireless Networks. In *1st Intl. Conf. on Mobile Computing and Networking*, Berkeley, CA, November 1995.

[B$^+$95b] B. Bershad et al. Extensibility, Safety and Performance in the SPIN Operating System. In *15th Symp. on Operating Systems Principles*, December 1995.

[BCZ96] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura. On Active Networking and Congestion. Technical Report CIT-CC-96/02, College of Computing, Georgia Institute of Technology, Spring 1996.

[Bor94] Nathaniel Borenstein. Email with a Mind of its Own: The Safe-Tcl Language for Enabled Mail. In *IFIP International Conference*, Barcelona, Spain, June 1994.

[Bra89] R. Braden, Ed. Requirements for Internet Hosts – Communication Layers. Request for Comments 1122, October 1989.

[Cla88] David D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *SIGCOMM '88*, pages 106–114, Stanford, CA, August 1988. ACM.

[CT90] D. D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *SIGCOMM '90*, Philadelphia, PA, September 1990. ACM.

[DH95] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. Request For Comments 1883, December 1995.

[E$^+$95] D. R. Engler et al. Exokernel: An operating system architecture for application-level resource management. In *15th Symp. on Operating Systems Principles*, December 1995.

[Gos95] J. Gosling. Java Intermediate Bytecodes. In *SIGPLAN Workshop on Intermediate Representations (IR95)*, pages 111–118, San Francisco, CA, January 1995. ACM. Appears in SIGPLAN Notices, 30, 3 (March 1995).

[Kle95] L. Kleinrock. Nomadic Computing (Keynote Address). In *1st Intl. Conf. on Mobile Computing and Networking*, Berkeley, CA, November 1995.

[KM87] C. Kent and J. Mogul. Fragmentation Considered Harmful. In *SIGCOMM '87*, Stowe, VT, August 1987. ACM.

[MP96] David Mosberger and Larry L. Peterson. Making Paths Explicit in the Scout Operating System. Technical Report TR 96-05, Dept. of Computer Science, University of Arizona, 1996.

[N$^+$96] B. Noble et al. Mobile Network Tracing (Draft RFC). Personal Communication, June 1996.

[Pos81] J. Postel, Ed. Internet Protocol: DARPA Internet Program Protocol Specification. Request For Comments 791, September 1981.

[SFG$^+$96] J. Smith, D. Farber, C. Gunter, S. Nettles, D. Feldmeier, and W. Sincoskie. SwitchWare: Accelerating Network Evolution (White Paper). Personal communication, June 1996.

[TGSK96] D. L. Tennenhouse, S. J. Garland, L. Shrira, and M. F. Kaashoek. From Internet to ActiveNet. Request for Comments, TNS Group, MIT LCS, January 1996.

[TW96] D. L. Tennenhouse and D. Wetherall. Towards an Active Network Architecture. In *Proc. of Multimedia Computing and Networking 96*, San Jose, CA, January 1996.

[WEK96] Deborah A. Wallach, Dawson R. Engler, and M. Frans Kaashoek. ASHs: Application-specific handlers for high-performance messaging. In *SIGCOMM '96*, Stanford, CA, August 1996. ACM.

[Yd96] Y. Yemini and S. da Silva. Towards Programmable Networks (White Paper). Personal communication, April 1996.