Experiences with Capsule-based Active Networking

Michael Hicks, Jonathan T. Moore, Scott Nettles, and David Wetherall

Abstract—
Active Networking adds programmability to the elements of the network, most aggressively by using programmable packets, or *capsules*. ANTS [1], [2] and PLANet [3], [4] are the most mature examples of capsule-based systems, both having been publicly available for several years. This paper presents our experience with these systems and the lessons they hold for the future of capsule-based Active Networking.

The paper focuses on four key issues: flexibility, performance, security, and usability. We consider how ANTS and PLANet address these issues, noting that despite substantial surface differences, both systems identify similar key problems and use closely related solutions. Based on our experience with these systems we conclude that capsule-based systems can achieve useful levels of flexibility, performance, and usability. Many aspects of security can also be adequately addressed, but some important problems related to denial of service remain as open problems.

I. Introduction

The IP-based Internet has been extraordinarily successful, due, in part, to its simple, 'one-size-fits-all' approach. The success of the Internet has driven application writers and service providers to demand new network-based applications, network services, and subnet technologies. Ironically, because of its simple approach, IP is unable to accommodate any significant degree of customization, and evolving the network service (e.g. evolving from IPv4 to IPv6) is slow and painstaking. This slow evolution is a product both of the speed with which standards are developed and approved, and the long delays in deploying new standards even after they have been agreed upon.

To redress poor customizability in the network, and the difficulties in rapidly evolving network services, researchers are developing *active* networks (AN). Active networks are networks whose elements are, in some way, programmable. This allows new code to be added both to support customized use of the network and to facilitate its evolution.

One can imagine many ways to add programmability to the network, ranging in level of ambition. On the less ambitious side, a router or firewall could present a management interface for configuring packet-processing; such mechanisms are available in most routers today [5]. More ambitious would be for routers to allow new code to be loaded to extend or modify packet processing [6], [7]. Perhaps most ambitious, we could allow programmability at the granularity of individual packets, flows of packets, or protocols. Here, we call these programmable packets *capsules*, but they are also sometimes refered to as *active packets*. Capsules may be executed on all or some of the nodes in the network, and this execution can control the routing (or even creation) of future capsules. Capsules may also be used to perform control operations within the network and, as such, may influence how other packets (capsules or not) are processed.

Michael Hicks, Computer Science, Cornell University, mhicks@cs.cornell.edu

Jonathan T. Moore, Computer and Information Science, The University of Pennsylvania, jonm@dsl.cis.upenn.edu

Scott Nettles, Electrical and Computer Engineering, The University of Texas at Austin, nettles@ece.utexas.edu

David Wetherall, Computer Science and Engineering, The University of Washington, djw@cs.washington.edu

Although we hold a broad interpretation of what constitutes AN—any combination of the above aspects, or other related programmability approaches, can and should be considered 'active'— this paper focuses on what we believe is the most radical and also most challenging approach to AN, the capsule approach. We aim to show that capsule-based systems can support a network infrastructure that is customizable by applications, and amenable to service evolution. Our consideration of capsules is based on our combined experience with the two most well developed AN systems supporting capsules, **ANTS** (**A**ctive **N**etworking **T**ransport **S**ystem) [1], [2], and **PLANet** (PLANet is the networking system for which **PLAN**, the **P**acket **L**anguage for **A**ctive **N**etworking, forms the capsule programming language) [4], [3]. Although these two systems were developed independently, and thus have some significant differences, what is truly striking is how in so many ways they attempt to address the same concerns and use closely related solutions.

To assess how ANTS and PLANet meet the goals of an active network, we first need a set of requirements. Here, we measure our systems according to the following criteria:

- *Flexibility*: How broad is the scope of programmability?
- *Security*: How are users and the network itself protected from harmful actions?
- *Performance*: How is performance affected by adding programmability?
- *Usability*: How easy is it to use and program the network?

Of course, a design choice that enhances one area, in particular flexibility, may unduly compromise other areas. At one end of the spectrum, the IP-based Internet provides little in the way of flexibility but provides reasonable levels of functionality in the other areas. The basic goal of any AN approach is to significantly improve flexibility without unduly compromising the areas of security, performance, and usability. The primary aim of this paper is to show that:

A capsule-based active network can provide a very significant increase in flexibility, without unacceptably compromising security, performance, or usability.

We begin in Section II by describing and analyzing the architecture of a capsule-based active network, using ANTS and PLANet as examples. In Section III, we present some applications developed using ANTS and PLANet and indicate how these applications benefited from the capsule approach. In Section IV, we look more closely at the design decisions that differentiate PLANet and ANTS, analyzing their tradeoffs. Finally, in Sections V and VI, we present open problems and general conclusions about the state-of-the-art in capsule-based active networks.

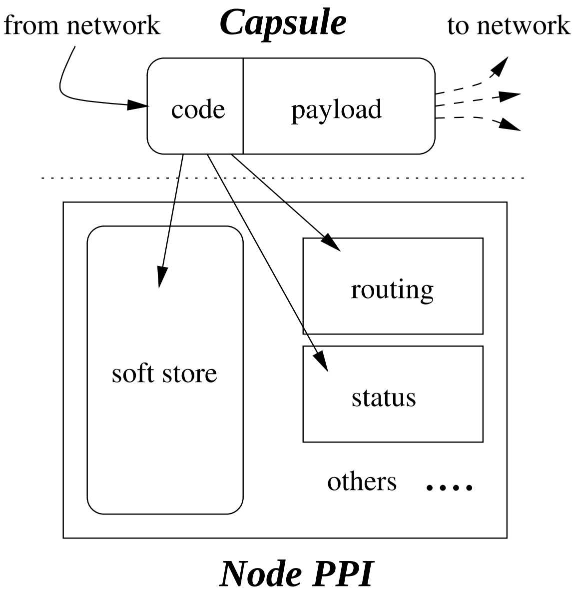

Fig. 1. Basic Capsule Architecture

II. Capsule-based Systems

In this section we present a general architecture for a capsule-based system. We then examine how ANTS and PLANet implement this architecture paying particular attention to the goals of flexibility, security, performance, and usability.

A. Capsule Architecture

Conceptually, a *capsule* is a packet that contains some code that directs how it is to be processed in the network. In both ANTS and PLANet, capsules are a central mechanism that enables widespread network programmability. User applications may customize network services by injecting capsules into the network to either forward, or alter the forwarding of, their data.

The basic capsule architecture is depicted in Figure 1. When a capsule arrives at node on which it requires processing, its code is evaluated. During evaluation, the capsule code may call functions present in the node's *packet programming interface*, or PPI. The PPI is essentially a library of services available to capsule programs. In both ANTS and PLANet, these services include a *soft store*, which allows capsules to store state on a node for short periods of time; *routing* tables, which present next hop information potentially gathered from a variety of algorithms; and *status* functions, which allow the capsule to query its own contents as well as information on node components, such as the length of the packet queue. As a result of capsule processing, a capsule may terminate (perhaps delivering data to some local application), forward itself onward in the network, or in the case of multicast-like services, spawn additional capsules.

PLANet and ANTS implement this architecture differently. These differences are captured in Table I. The table breaks down the two components of the capsule architecture, the capsule and the node PPI, and presents the design decisions made by both systems. In the discussion below, we highlight the motivations for some of these differences in meeting the overall goals of AN. We defer discussion of some of the more interesting differences until Section IV.

Flexibility Capsules provide a simple, elegant way to customize packet processing in the network: applications simply include the necessary processing code with their packets. A number of AN applications we have built, or are building, are presented in the next section to illustrate how this is useful.

ANTS capsule programs consist of Java Virtual Machine (JVM) code, and are executed within a restricted environment for security reasons. In PLANet, all capsule programs are written in a special-purpose programming language, PLAN [4]. PLAN was designed to be expressive enough to write interesting programs, but limited enough to prevent harmful programs; e.g. all PLAN programs are guaranteed to terminate.

In both systems the Node PPI is trusted at some level, and, for maximum flexibility, is implemented in a general-purpose (but type-safe) language (Java for ANTS, and OCaml [8], an ML dialect, for PLANet[1]). In this way, the more expression-limited packet programs act as scripts that glue together node-resident services. This architecture allows users to *customize* the processing of their packets, but leaves most of the task of network service *evolution* to trusted Node PPI writers.

In ANTS, capsule programs are evaluated on each hop *en route* to the packet's ultimate destination. In PLANet, however, packet code does not have to be evaluated on each hop. Instead, the packet may make use of a node-resident *routing function*, referred to in the packet, to take care of basic transport. In effect, this allows PLANet capsules to have two programs: one program resides in the packet and is evaluated at selected points throughout the network, while the other resides on the node and is evaluated on each hop between those selected points, but only provides routing.

Security In the vision of both ANTS and PLANet, *untrusted users* may program and inject capsules into the network. While this opens up the network to everyone, improving system flexibility and usability, it could weaken node security. In particular, if capsules may be written in general-purpose languages, then malicious users can mount denial-of-service attacks via infinite loops, or, in unsafe languages like C, can compromise system integrity by dereferencing null pointers. For this reason, both ANTS and PLAN compromise flexibility in favor of security to protect the resources on each network node and the network resources in aggregate.

Node resources are protected in several ways. First, both systems use type-safe languages for programming capsules. This prevents type-errors, buffer over-runs, and storage management errors from compromising basic system integrity. Similar provisions are made to prevent different users' capsules from directly interfering with one another, or from accessing protected portions of the system state. Next, both systems restrict the expressiveness of capsules to protect nodes from malicious or buggy programs. In ANTS, capsule programs are monitored and killed if they exceed certain CPU and memory usage thresholds, while in PLAN programs are by nature guaranteed to terminate. Finally, capsule programs have access to a limited PPI; e.g. only

[1] A more recent implementation of PLANet uses a type-safe, C-like language called Popcorn [9].

	ANTS		PLANet	
	Capsule	Node PPI	Capsule	Node PPI
Language	Java	Java	PLAN	OCaml
Code location	code cache	on node	in packet	on node
Expressibility	restricted	general purpose	limited	general purpose
Authentication?	no	no	no	when needed
Extensible?	*n/a*	no	*n/a*	yes

TABLE I
ANTS AND PLANET CAPSULE IMPLEMENTATIONS

a subset of the standard Java libraries is available to ANTS programs.

To prevent the overuse of network resources, both ANTS and PLANet use a generalization of IP's Time-to-Live (TTL) counter. When new capsules are created during capsule execution, the TTL of the new capsules must be less than that of the current capsule, guaranteeing that capsules cannot roam about the network forever.

Two key aspects of the PLANet Node PPI are that it is *extensible*—it may be extended with new functionality at runtime—and that it is *protected*—while some services are available to all capsules, some are available only to trusted capsules. The first property allows the PPI to evolve as the need for new services arises. The second property, while useful in itself, necessarily follows from the first: if general-purpose code can be installed on the node, there must be some way to secure that installation [10]. While its current implementation does not support these features, ANTS could be extended to include them.

Performance A drawback of the basic capsule architecture is that capsules carry their code with them, consuming space in the packet and thus reducing the amount of useful payload. In ANTS, capsules do not actually carry their code, but instead *refer* to it with a special identifier, formed by taking the MD5 [11] fingerprint of the associated capsule code. When a capsule arrives at a node for processing, the node looks up the identifier in its *code cache* of capsule programs to extract the packet's program. If the program is not present in the cache, it must be retrieved. The current approach is to query the previous node for the code, and retrieve it from there if present. If that node does not have the code, the packet is dropped. This scheme allows code to originate with an end-user, and then inductively proliferate along application paths through the network. While PLANet currently does not support caching, it could be added naturally.

Usability Language features of PLAN and Java help make the network that uses them for its capsule programming language more usable. In particular, both PLAN and Java are statically and strongly typed languages. This means that users know *before* they send capsules into the network that there is no possibility of a runtime type error, which should simplify the debugging process. Similarly, PLAN's termination guarantee means that there is no way for a user to inadvertently write a program that infinitely loops (which would result in its being abruptly killed by the system).

There are interesting tradeoffs in some of the decisions made in the design of ANTS and PLANet, especially in the areas of security and performance. We defer discussion of these design decisions to Section IV, focusing next on the variety of applications that have been possible in PLANet and ANTS.

III. Application Experience

Both systems have been publicly available for several years [2], giving us and others significant experience with actually using capsules. In this section, we turn to this experience to show some of the wide range of network applications capsules can support. We present a number of examples, from network management to firewalls to packet forwarding optimizations to full internetworks.

A. Network management

Managing and diagnosing large networks is an important challenge. Current techniques (e.g., SNMP [12]) provide only very basic, low-level functionality. Diagnosing and correcting problems can be very tedious and time consuming, requiring many round trips between the problem node(s) and the network operations center.

Capsules allow packets implementing simple decision making capabilities to be deployed directly to the problem spots. As such, the administrator can use more high-level and sophisticated query and correction operations directly, without round trips, and such techniques can either come prepackaged, or can be created on-the-fly to help solve some unanticipated problem. Furthermore, a capsule can travel between several nodes, aiding in diagnosing and correcting problems of a distributed nature.

Several AN projects have focused on this domain, particularly the SmartPackets project at BBN [13]. PLAN has been used for configuring and managing Cisco routers [14] and as a front-end for lower-level capsule-based network management.

B. Application-specific routing

As noted, IP's one-size-fits-all approach has been key to its success. However, in the modern Internet many applications make a wide variety of different demands: high bandwidth (file transfer), low latency (real-time games), low jitter (multimedia

[2]For ANTS, see `http://www.cs.washington.edu/research/networking/ants/`, and for PLANet, see `http://www.cis.upenn.edu/~switchware/PLAN/`.

streaming), and so on. It is unclear whether one size can really fit all.

In many cases, the service required will determine the routing needed. Unfortunately, in today's network routing is basically a black-box, and routing decisions are made primarily on the basis of a most rudimentary criteria, hop-count.

Capsules can allow routing decisions to be made by applications, based on specific needs. One approach is for control capsules to determine the routing for data capsules. For example, we have used a control capsule to search a network for a path suiting requirements, and to then set up a special flow route for data packets to follow [3]. We have also explored a related approach where control capsules detect nearby web caches that may not lie directly on the path between client and server and divert web requests without requiring server redirects [15].

C. Application-specific forwarding

The previous examples exploit programmability in the control plane. Even in the routing example, *how* data packets are forwarded has not been customized, just the way in which their routes are determined. A much more aggressive approach inserts application-specific processing into the forwarding loop of network nodes.

One use of this approach is to selectively forward at a congested node. For example, a video stream composed of MPEG I, P, and B frames can drop the less-important P and B frames in favor of the vital I-frames. Furthermore, if part of a frame has already been dropped, the rest of the frame can be dropped as well, since the partial frame will be useless and there will not be time for retransmission.

It is also possible to push some processing out into the network to relieve the load on a popular web server. For example, an on-line auction could benefit by pushing the current highest bid for an item outwards. Incoming bid capsules could examine this state and then preemptively reject themselves if they contain bids that are too low. This would result both in a reduction in server load from the elimination of spurious bids as well as faster client response for bids which were not successful.

A particularly aggressive application of active forwarding would be capsules that adapted themselves at boundaries between fundamentally different networks. For example, capsules could compress themselves at a dial-in server before crossing serial links, particularly for slow-speed modem connections where there are sufficient CPU cycles to support this. Similarly, capsules could add forward-error-correction information to themselves before crossing lossy links [16]. It might also be feasible to have a multicast data packet re-encode itself into multiple video layers at routers connecting networks of differing speeds.

D. PLANet the network

All of the previous examples will work when the network is partially capsule-based where some of the routers can not execute capsules and some packets do not contain code. ANTS has focused on such a world, and is implemented as an overlay on top of the current Internet. A significant advantage of this approach is that it helps us to understand how capsules might be incrementally deployed into the existing network.

PLANet has taken a complementary approach in the form of the PLANet network. PLANet is a pure active internetwork: every router is active and supports capsule execution and dynamic extensions to its PPI, and every packet in the network is a capsule containing PLAN code. All networking operations, including data transport, route updates, and address resolution, work by sending and executing capsules. New services can be created simply by coding new capsule programs and if needed by extending the PPI over the network.

Although such a radical design certainly will not replace the IP-based Internet in the near future (if ever), it does demonstrate an important point about capsule-based AN: purely capsule-based systems are powerful enough to build even the most complex network software.

E. Chunks

Building PLANet raised an important issue, the need for an internetwork to support both fragmentation and encapsulation of packets. Since PLANet only had capsules, we needed to be able to treat capsules both as abstract data to be encapsulated, and as a concrete representation that could be fragmented (or checksummed, encrypted, etc.). This was achieved by introducing *chunks*, which are capsules represented as PLAN data. Chunks can be used as arguments to capsules, they can be marshalled into byte-arrays, and they can be (explicitly) executed.

As an example, consider fragmentation. When a capsule is bigger than the MTU of some subnet, its chunk representation is converted into a byte-array. This byte-array is broken into sufficiently small pieces and sent as data inside of other capsules. These capsules contain code that inserts the fragments into a reassembly table. Once all the fragments arrive, the chunk is reassembled and then evaluated, just as if the original capsule had been sent whole across the network [17].

F. Active Firewalls

Capsules have been used to implement firewalls that go significantly beyond the basic filtering done today [10]. The idea is that rather than permit or deny packets from entering the trusted network, the firewall encapsulates (using chunks perhaps) the incoming 'guest' packet in a special 'escort' capsule and sends it on to its destination. When the escort capsule executes on the destination, it sets up an execution environment that allows the guest packet to only access functionality that is in keeping with its level of privilege. For example, the guest may be allowed a to write, but only if the write goes to a screen and not to the file-system.

IV. Comparing Design Decisions

In this section, we elaborate further on our experience with PLANet and ANTS, and, in particular, on some of the design decisions made in each system, and how they impact the overall effectiveness of each system. We consider the benefits and drawbacks of caching capsule programs, means to secure the network from attack, and the ramifications of the choice of capsule language.

A. Code Caching

Both ANTS and PLANet follow the same conceptual model for capsules in which a packet carries its own processing code. The systems differ, however, in that PLANet packets actually contain PLAN code, whereas ANTS capsules contain a *reference* to their protocol code. These choices target particular kinds of applications with complementary advantages and disadvantages.

Caching is motivated by the idea that customization is more useful for application *flows* than for individual packets. Flows are characterized by streams of packets that require the same processing code at each node. In this case, caching the code for packets at each node yields savings in network usage (since the code is no longer carried in the packet) and potentially in packet evaluation (in the case that the packet code requires some preparation, e.g. unmarshalling, before it can be evaluated), while the additional overhead of loading and saving the code is amortized over the lifetime of the flow. Cached code can also be optimized, for example with a JIT compiler, thus paying some initial performance cost for a longer term performance gain. Code caching in ANTS is especially important because capsule code is written in Java, whose programs tend to be lengthy (relative to PLAN packets, at least) and have a high startup cost due to verification, dynamic linking, and sometimes just-in-time compilation.

When the network becomes highly loaded with capsules of different flows, there will be contention for cache space. In the worst case, each time some capsule code is inserted in the cache it is immediately evicted by the code of a different capsule type. This results in all capsules having to continuously load their code, taxing the network's resources with unproductive work. We expect that this problem will be mitigated by the fact that though many new services might be tried over time, a small number of services will account for virtually all of the traffic at any given instant, because of the existence of higher layer flows and a dependence on third party software. Of course, there is still the possibility of a denial-of-service attack against the cache system that would create contention artificially. Until we have more widely-deployed active networks, it will be hard to tell whether cache contention will be a problem in practice.

There are also interesting applications that are not flow-based and thus do not benefit from caching. For example, caching code for error-reporting capsules (akin to ICMP packets) seems not to be useful, since they arise infrequently and are not grouped in flows. A similar argument may be made for capsules involved in network management. We might also imagine special capsules that are used to set up longer running application flows. These capsules are used once and then never again, and so it seems unnecessary to have to pay for caching.

Some of the benefits of caching may be obtained by other means. For example, a well-chosen wire format for capsule programs may decrease the size of programs as well as their startup cost. Indeed, some of our recent research [18], discussed briefly below, suggests that simple data transport packets can be encoded in a few tens of bytes of code and can be executed as quickly as a software IP router[3]. This means that carrying code in the capsules themselves is not prohibitive to performance and thus execution need not rely on the freshness of a code cache.

On the whole we feel that many applications will benefit from caching. We expect that widely deployed capsule based systems will support code transmission both by value when caching is not useful and by reference when caching is useful.

[3]This should not be surprising; IP headers can be viewed as small programs in a very limited programming language.

B. Node Protection

The goal of providing network robustness in the face of additional programmability is an important one in both systems. In particular, we want to prevent capsules from damaging or crashing a node, or from using an undue amount of that node's resources.

Node protection is provided in ANTS through the use of the Java language; this provides memory protection via static verification, thus keeping capsules from referencing critical portions of the node's memory. Similarly, PLANet's PLAN interpreter enforces memory protection using runtime tests. Additional functions performed by the capsules are mediated by the node's PPI.

The two systems differ, however, in how they limit a capsule's resource usage on a particular node. ANTS uses watchdog timers and hard memory limits to kill capsules that use too much CPU time or allocate too much storage. This approach is straightforward, and imposes a hard limit on the amount of resources consumed per capsule, simplifying considerations of scalability. On the other hand, this approach may impact performance, since accounting must be done in the runtime's thread system and memory allocation routines [19]. Furthermore, depending upon how tight the usage bounds are, it may be difficult for capsule programmers to know that their code will successfully complete within the given limits. This problem is only exacerbated by the (likely) prospect of different limits for different administrative domains.

PLANet, on the other hand, relies upon the limited expressiveness of the PLAN language to bound per-node resource usage. PLAN programs are guaranteed to terminate, and with some slight modifications [4], [18], this guarantee can be tightened further so that all packets consume resources within a constant factor of their length (this is the same property that vanilla IP packets have). Such a bound obviates the need for watchdog timers or memory limits, as a node can know *a priori* that a PLAN capsule will behave reasonably. As a result, programmers know that their programs will not be arbitrarily truncated. These guarantees do not come for free, however; the price is in reduced capsule expressibility.

C. Language Choices

One of the obvious differences between ANTS and PLANet is the choice of capsule programming languages. There have been advantages and disadvantages ANTS's use of a restricted subset of a general-purpose language as well as PLANet's use of a domain-specific language.

Since both ANTS capsules and its PPI are written in Java, other researchers have found the ANTS toolkit to be very accessible. The main drawbacks have come from the fact that Java is a general-purpose language but ANTS capsules use a restricted subset of the language for security purposes. In particular, performance has suffered due to the use of the full-featured JDK runtime system. It is also somewhat difficult to determine that the set of class libraries to which capsules have access are 'safe.' On the other hand, the furious pace of improvements to Java is obtained 'for free.'

PLANet uses a domain-specific capsule language, PLAN, for its packet programs, and a functional language, OCaml [8], to implement its node PPI. The benefits of these decisions are essentially a mirror-image of the drawbacks of ANTS' use of Java. Because PLAN was designed as a limited language, it is straightforward to provide specialized, efficient runtime support. Furthermore, as mentioned earlier, some local node resource usage properties are obtained through the design of the language itself. On the other hand, PLANet's node implementation in OCaml and unique packet language have made it less attractive to other researchers interested in quickly prototyping new active applications. And while the OCaml language is currently supported and oft-improving, PLAN itself only improves when its maintainers tune its interpreter.

In the long run, it seems likely that performance and security will be the driving forces behind the choice of capsule language in a widely-deployed active network. Programming language technology is a convenient and powerful tool for governing resource usage [18], [20] and providing node protection [21], [22], and progress is continuing on making such techniques efficient. It seems certain that neither Java nor PLAN are the last word on capsule programming languages.

V. Open Problems

Both systems have significant room for improvement, in part to incorporate useful aspects of the other. For example, PLANet would be enhanced by adding code caching, while ANTS would become more expressive by adding features such as chunks. However, here, rather than looking at issues involving each individual system, we will consider our view of the open issues for capsule-based systems in general.

Of our goals, despite the restrictions placed on them, both systems provide significant flexibility, supporting both customizability and evolution. The ability to build a number of diverse networking applications shows that our systems are usable and useful. However, questions remain concerning their meeting the goals of performance and security.

A. Performance

It is useful to consider various aspects of the network when asking if it has been demonstrated that capsules can provide acceptable performance. For control plane operations, as well as other operations that are off the packet forwarding critical path, the current prototypes provide more than adequate performance. Since the flexibility of capsules can be taken significant advantage of in these areas, we believe that capsule-based AN can find an application here, even if it never becomes a part of packet forwarding.

The application of capsules to packet forwarding is a question of what the ratio of computational resources is to the possible packet rate and bandwidth. The prototype systems discussed here are already suitable for links in the tens of Mbps range, while more advanced systems [23], [18], perform similarly to other software-based routers running on general-purpose hardware, where hundreds of Mbps can be supported. Since capsules can provide sophisticated firewall capabilities, it is important to note that these results suggest that current firewalls could be made active without a significant impact on performance. Furthermore, routers with a high aggregate bandwidth are often built by using per port line cards that then switch over a high-bandwidth switching fabric. Although no prototypes yet exist, we believe that by adding processing power to each line card, capsules could also be used in this architecture. This is because, by design, capsules are independently processed. If our belief is correct, capsules can be employed in routers with many Gbps of aggregate bandwidth, as long as individual ports are no faster than those supported by software-based routers.

The remaining nodes are routers that provide gigabit (or more) bandwidth on individual ports, typically the routers at the network core. Although we are optimistic about eventually supporting capsules in such core routers, we also believe this will be a significant engineering challenge. The implication of this is that for the foreseeable future capsule-based routers will need to co-exist with more conventional routers. Since deployment of AN technology will happen only gradually, this restriction is one we will have to accommodate in any case. Fortunately, the current prototypes show that capsule-based systems can co-exist with conventional routers, so we conclude that performance should not be an impediment to deploying these systems in much of the network.

B. Security

Security has been a key concern for AN in general, and for both our systems in particular. Security can be compromised in two ways. First, a capsule could cause an illegal operation to occur on another packet or the node itself, possibly resulting in failure; this aspect of security is termed *safety*. Second, a capsule could inappropriately consume node resources, resulting in a denial of service.

Both systems show that capsules can be made secure with respect to safety. Although details vary, both systems do this by providing well-defined execution environments for capsules, and by carefully controlling how capsules interact with routers and other capsules. Furthermore, because this issue is important to the entire mobile code/extensible systems community, we expect significant new results and techniques to further enhance our ability to provide safety and safety-based security.

Not surprisingly, denial-of-service attacks are harder to deal with. There are a number of places where denial of service may occur: at individual nodes, inside the network, and at the edges of the network. Node-centered attacks can arise if a capsule is allowed to use unbounded resources at an individual node. We believe both systems adequately deal with these problems by bounding resource usage in some manner. Attacks from the edge of the network are harder to deal with. The existing network deals with these only crudely, in that only limited bandwidth is available to each user to employ such attacks, and in that such attacks can be traced and the attackers held responsible. Unfortunately, none of the capsule-based systems built so far offer any better solutions. Fortunately, the solutions are also no worse. We believe that this is an important avenue of investigation, not just for capsule-based systems, but for packet switching in general. There is also hope that the flexibility of active networks may *improve* our ability to secure against such attacks.

The final problem is that capsules may use limited resources to enter the network and on individual routers but still cause unacceptably large resource usage within the network as a whole. This is because capsules may fan-out and if this fan-out is not controlled it may be essentially unbounded. PLANet attempts to deal with this issue by forcing the TTL field of a capsule to bound the TTL of all child packets. This works, but leaves the difficult problem of how to allocate and manage the TTL. ANTS attempts to deal with the problem by certifying capsule programs. Such certification may still allow faulty programs, and certainly will slow the speed with which new protocols can be employed, but it will reduce the problem and add an avenue for accountability. Another solution would be to not allow packets to fan-out. This is workable for some applications, but is a serious limitation in others, such as multicast. We do not consider any of these solutions to be satisfactory, and we believe this issue is the most important unresolved technical problem inhibiting the deployment of capsules in the public network. We are just beginning to work towards developing new approaches to address these issues.

SNAP

Although a full discussion is outside the scope of this paper, we have made significant progress in addressing these open issues in our second generation capsule language, SNAP (Safe and Nimble Active Packets [18]). SNAP's design addresses the issue of resource usage security by introducing a new model of capsule execution. The SNAP model restricts capsules to using time, space, and bandwidth linear in the length of the packet. This not only significantly bounds resource usage, but, more importantly, makes it predictable. Among the advantages of this is making enforcing resource usage policies significantly easier than with ANTS or PLANet. This model is enforced by a combination of SNAP's design and its implementation.

SNAP also improves on the performance of ANTS and PLANet. SNAP itself is a very simple byte-code language, which admits a simple and efficient implementation. Furthermore SNAP is designed to be executed 'in place,' meaning that SNAP programs can be executed exactly as they are when they come off the wire, avoiding the cost of marshaling and unmarshalling capsules before execution. SNAP has been implemented as a router alert option for IP in the Linux kernel, and this implementation performs only marginally worse than the Linux IP implementation (being able to forward 100+ Mbps), despite the need to execute SNAP code on every hop.

VI. Conclusions

Both ANTS and PLANet support a very powerful and sophisticated capsule architecture for active networking. Despite superficial differences, both systems focus on a similar set of design issues and have reached similar solutions. Because ANTS and PLANet have demonstrated their usefulness by building quite different types of networking services, together they show that capsules have a wide applicability, both as part of an overlay in the existing network, or as the fundamental basis of some future network.

At the beginning, we asked if capsule-based systems could provide sufficient flexibility, performance, usability, and security to be widely useful. Our experience strongly suggests that for the first three issues, the answer is "yes". Our experience also shows that many of the security issues can also be dealt with, but that some important issues still remain open to further research.

References

[1] David J. Wetherall, John Guttag, and David L. Tennenhouse, "ANTS: A toolkit for building and dynamically deploying network protocols," in *IEEE OPENARCH*, April 1998.

[2] David Wetherall, "Active network vision and reality: lessons from a capsule-based system," in *17th Symp. on Operating Systems Principles (SOSP'99)*, Kiawah Island, SC, December 1999, ACM, pp. 64–79.

[3] Michael Hicks, Jonathan T. Moore, D. Scott Alexander, Carl A. Gunter, and Scott Nettles, "PLANet: An active internetwork," in *Proceedings of the Eighteenth IEEE Computer and Communication Society INFOCOM Conference*. March 1999, pp. 1124–1133, IEEE.

[4] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles, "PLAN: A packet language for active networks," in *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*. September 1998, pp. 86–93, ACM.

[5] Robert Ciampa, "Flexible intelligent routing engine (FIRE)," Technical report, 3COM, 1999.

[6] D. Scott Alexander, Marianne Shaw, Scott M. Nettles, and Jonathan M. Smith, "Active bridging," in *Proceedings, 1997 SIGCOMM Conference*. ACM, 1997.

[7] Yechim Yemini and Sushil da Silva, "Towards programmable networks," in *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, September 1996.

[8] Xavier Leroy, *The Objective Caml System, Release 3.04*, Institut National de Recherche en Informatique et Automatique (INRIA), 2002, Available at `http://caml.inria.fr/`.

[9] Greg Morrisett, Karl Crary, Neal Glew, Dan Grossman, Richard Samuels, Frederick Smith, David Walker, Stephanie Weirich, and Steve Zdancewic, "TALx86: A realistic typed assembly language," in *Second Workshop on Compiler Support for System Software*, Atlanta, May 1999.

[10] Michael Hicks and Angelos D. Keromytis, "A secure PLAN," in *Proceedings of the First International Workshop on Active Networks*, Stefan Covaci, Ed. June 1999, vol. 1653 of *Lecture Notes in Computer Science*, pp. 307–314, Springer-Verlag.

[11] R. Rivest, "The MD5 message-digest algorithm," Request For Comments 1321, IETF, April 1992.

[12] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," Tech. Rep., IETF RFC 1157, May 1990.

[13] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, R. Rockwell, and C. Partridge, "Smart packets for active networks," in *Proceedings of the 1999 IEEE 2nd Conference on Open Architectures and Network Programming (OPENARCH'99)*, March 1999.

[14] Osman Ertugay, Michael Hicks, Jessica Kornblum, , and Jonathan Smith, "Agents in network management," Unpublished manuscript, April 2000.

[15] Ulana Legedza and John Guttag, "Using network-level support to improve cache routing," in *Proceedings of the 3rd International WWW Caching Workshop*, June 1998.

[16] I. Hadzic, W. S. Marcus, and J. M. Smith, "On-the-fly programmable hardware for networks," in *Globecom'98*, 1998.

[17] Jonathan T. Moore, Michael Hicks, and Scott M. Nettles, "Chunks in PLAN: Language support for programs as packets," in *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing*, September 1999.

[18] Jonathan T. Moore, Michael Hicks, and Scott Nettles, "Practical programmable packets," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, April 2001.

[19] Michael Hicks, "PLAN system security," Technical Report MS-CIS-98-25, Department of Computer and Information Science, University of Pennsylvania, April 1998.

[20] Karl Crary and Stephanie Weirich, "Resource bound certification," in *Proceedings of the Twenty-Seventh Symposium on Principles of Programming Languages*, January 2000, To Appear.

[21] R. Wahbe, S. Lucco, T. Anderson, and S. Graham, "Efficient software-based fault isolation," in *Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP)*, December 1993.

[22] George Necula, "Proof-carrying code," in *Proccedings of the 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'97)*, January 1997.

[23] E. Nygren, S. Garland, and M. F. Kaashoek, "PAN: A high-performance active network node supporting multiple mobile code systems," in *Proceedings of the 1999 IEEE Second Conference on Open Architectures and Network Programming (OPENARCH'99)*, March 1999.