Evolution in Action: Using Active Networking to Evolve Network Support for Mobility

Seong-Kyu Song[1], Stephen Shannon[1], Michael Hicks[2], and Scott Nettles[1]*

[1] Electrical and Computer Engineering Department
The University of Texas at Austin
{sksong,shannon,nettles}@ece.utexas.edu
[2] Department of Computer Science
University of Maryland, College Park
mwh@cs.umd.edu

Abstract. A key early objective of Active Networking (AN) was to support on-the-fly network evolution. Although AN has been used relatively extensively to build application-customized protocols and even whole networking systems, demonstrations of evolution have been limited.

This paper examines three AN mechanisms and how they enable evolution: active packets and plug-in extensions, well-known to the AN community, and update extensions, which are novel to AN. We devote our presentation to a series of demonstrations of how each type of evolution can be applied to the problem of adding support for mobility to a network. This represents the most large-scale demonstration of AN evolution to date. These demonstrations show what previous AN research has not: that AN technology can, in fact, support very significant changes to the network, even while the network is operational.

1 Introduction

The early promise of Active Networking (AN) was two-fold: to introduce computation in the network both for the purpose of application-specific customization and to increase the pace of network service evolution [33, 4]. Thus far there have been numerous demonstrations using AN to customize networks for a specific application's needs (e.g. [21, 10, 22]) as well as some demonstrations of the use of AN to build network infrastructures from scratch [10]. However, while the key mechanisms are in place in many AN systems, there have been few demonstrations of using AN to facilitate network service evolution. One example is the Active Bridge [2], which demonstrated the use of AN to switch between bridging protocols dynamically. Unfortunately, this upgrade was quite simple and did not demonstrate the overall scope of AN possibilities.

* This material is based upon work supported by the National Science Foundation under Grant No. CSE-0081360. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

J. Sterbenz et al. (Eds.): IWAN 2002, LNCS 2546, pp. 146–161, 2002.

We hope to fill this gap in the research record by showing how AN can be used to upgrade a network's services on the fly, without centralized coordination and without halting network service. By doing so, we are making a strong claim that AN *can* be used to quicken the pace of network service evolution. For our demonstration, we present two main examples. First, we show how to augment an active network that provides standard, IP-like service to support routing for mobile hosts, in the spirit of Mobile-IP [29, 30]. Second, we show how to augment an active, mobile ad-hoc network with support for multi-hop routing. While our primary goal is to demonstrate the capabilities of AN technology in evolving a network, a secondary goal is to explore the suitability and usefulness of AN techniques within the mobile networking domain.

These demonstrations have had the side-benefit of helping us better understand the types of evolution that AN's can and should support, and the mechanisms required to implement them. In particular, we have identified three programmability mechanisms—active packets (APs), plug-in extensions, and update extensions—which in turn enable three different types of evolution. APs and plug-in extensions have been well studied in AN research (e.g. [32, 36, 8, 28, 35, 14, 24, 3, 37, 20, 5, 6]) but update extensions have not been explored for AN [12].

A key differentiating factor among these mechanisms concerns whether a new service is application-aware or application-transparent. Application-aware network services require that for an application to use a new service, it must be aware that it is doing so. For example, using IP-style multicast requires the sending application (or perhaps the middle-ware used by the application) to send to a special multicast address. In contrast, application-transparent services are those that act without the application's knowledge. For example, in IP-style mobility, packets destined for a host's home network are transparently forwarded to that host's current remote network; the sending application does not need to be aware of mobile IP services for them to work. In making this distinction, we have realized that APs and, in many cases, plug-in extensions cannot solely provide transparent service; they require the aid of update extensions. However, the added power of plug-in and update extensions makes them a greater security risk, implying that services would benefit from using a combination of mechanisms to balance the needs of the application and of the network.

We hope that this paper will raise the level of discussion on the mechanisms required by active networks to truly support not just application-level customization, but true network service evolution. We begin in Section 2 by describing the three types of network evolution and their enabling AN technologies. In Section 3 we discuss our implementation testbed, MANE, and its programmability mechanisms. We also provide enough background on mobile-IP-style mobility and ad-hoc networks for the reader to understand our demonstrations. In Sections 4, 5 and 6 we subsequently present our demonstrations of each of the three types of network evolutions described in Section 2. Finally, in Section 7, we summarize some of the lessons learned from our case studies and conclude.

2 Network Evolution

Broadly speaking, by "network evolution" we mean any incremental change to a network that modifies or enhances existing functionality or adds new functionality. In the context of Active Networking a somewhat more ambitious goal is appropriate: evolution should be able to occur at remote nodes while the network is operational with only minimal disruption to existing services.

AN achieves evolution by changing the programs that operate the network. Thus the ways in which we can evolve the network are dictated by the programmability mechanisms that are available to make such changes. In some cases, these mechanisms are AN specific, but generally they are drawn from general programming language technology. Thus, although later we will choose instances of these mechanisms that are specific to our platform, this discussion is general and applies broadly to AN systems.

In this section, we describe three mechanisms for achieving AN evolution. In each case, we discuss what type of evolution is supported by the mechanism. We also consider how the mechanism might support application aware or transparent evolutions. Later in the paper we will show concrete examples of evolutions of each type using the mechanisms outlined here.

2.1 Active Packets

Active packets (AP) are perhaps the most radical AN technology for evolution and they are the only mechanism that, at a high-level, are specific to AN. Such packets carry (or literally are) programs that execute as they pass through the nodes of network. A packet can perform management actions on the nodes, effect its own routing, or form the basis of larger protocols between distributed nodes, e.g. routing protocols. Such packets can form the glue of the network, much like conventional packets, but with qualitatively more power and flexibility.

The AN community has explored a number of AP systems. The early systems include Smart Packets [32], ANTS [36], and PLAN [8], while more recent systems include PAN [28], SafetyNet [35], StreamCode [14], and SNAP [24]. Although these systems differ on many details of their design and implementation, they all support the basic AP model and thus the same general styles of evolution.

Active packets support the first and simplest type of network evolution we identify, *Active Packet evolution*, which does not require changes to the nodes of the network. Instead, it functions solely by the execution of APs utilizing standard services. The disadvantage of this approach is that taking advantage of new functionality requires the use of new packet programs. This means that at some level the applications using the functionality must be aware that the new functionality exists. This is the kind of evolution facilitated by pure AP systems, such as ANTS [36] and in essence it embodies the AN goal of application-level customization. We will demonstrate how this kind of evolution can be used to implement application-aware mobile-IP style mobility in Section 4.

2.2 Plug-In Extensions

The programmability mechanism that is broadly familiar outside the AN community is the *plug-in extension*. Plug-in extensions can be downloaded and dynamically linked into a node to add new node-level functionality. For this new functionality to be used, it must be callable from some prebuilt and known interface. For example, a packet program will have a standard way of calling node resident services. If it is possible to add a plug-in extension to the set of callable services (typically by extending the service name space) then such an extension "plugs in" to the service call interface.

Plug-in extensions are commonly used outside of AN. For example, the Linux kernel enables plug-in extensions for network-level protocol handlers, drivers, and more. Java-enabled web browsers support applets, which are a form of plug-in extension. Plug-ins are also common to AN. In CANES [3], nodes execute programs that consist of a fixed underlying program and a variable part, called the injected program. The fixed program contains *slots* that are filled in plug-in extensions. In Netscript [37], programming takes place by composing components into a custom dataflow. In this case, each element in the composed program is a plug-in, and the abstract description of such an element forms the extension interface. Plug-ins are used in hardware-based approaches as well, including the VERA extensible router at Princeton [20], and Active Network Nodes (ANN) at Washington University and ETH [5, 6].

Plug-in extensions support the second type of evolution we identify, *plug-in extension evolution*. When used in conjunction with APs, packet programs can use new node resident services specialized to their needs rather than just standard services. Such evolution is particularly important if standard services are not sufficient to express a needed application. The combination of Active Packet and plug-in extension evolution is facilitated by systems such as PLANet [10], ALIEN [1], and SENCOMM [17]. We will demonstrate how this kind of evolution can be used to add multi-hop routing to an ad-hoc mobile network in Section 5.

Plug-in extensions that must be referenced by new AP programs are obviously not application transparent. However, as long as a plug-in simply replaces an existing interface, whether that interface is accessed from an AP or even in a more conventional system that does not support APs at all, then the evolution can be application transparent. This situation in occurs in CANES, for example. However, the system still must have been designed to allow the required change (e.g. in CANES, this is made possible by the slots in the fixed program).

2.3 Update Extensions

The final programmability mechanism we consider is the *update extension*. Update extensions may also be downloaded, but they go beyond plug-in extensions in that they can update or modify existing code and can do so even while the node remains operational. Thus, such extension can add to or modify a system's functionality even when there does not exist an interface for it to hook into.

There is significant research literature on such extensions (e.g. [7, 23, 15] to name a few) although in general the focus has been on code maintenance rather than evolving distributed system functionality. The specific system we are using [11] was initially inspired by the difficulties of crafting a plug-in interface for the packet scheduler in PLANet [10, 12]. However, the system itself is not specific or specialized to AN.

Update extensions support the final type of network evolution we identify, *update extension evolution*, which occurs when network nodes are updated in more or less arbitrary ways. This means that the evolution can affect the operation of existing functionality, even if such functionality was not explicitly designed to be extended (as was required for plug-in extensions). This means that in general evolutions that are transparent to the clients of a service are feasible. To our knowledge, only our current system, MANE, supports this type of network evolution. We will demonstrate how this kind of evolution can be used to implement application-transparent, mobile-IP-style mobility in Section 6.

3 Technology

Before presenting our examples, we discuss the key technologies used in their implementation. We discuss our Mobile Active Network Environment (MANE), and the details of how it provides the key evolutionary mechanisms, as well as the networking protocols we will evolve MANE to support.

3.1 MANE

MANE is the logical descendant of our previous AN testbed, PLANet [10], with improvements in a number of areas. MANE is very loosely based on the limited initial prototype described by Hornof [16].

MANE implements all three evolutionary mechanisms. AP programs are written in a special-purpose language, PLAN [8]. MANE nodes and their extensions (both plug-in and update) are implemented in software based on Typed Assembly Language (TAL) [26]. TAL is a cousin of proof-carrying code (PCC) [27], a framework in which native machine code is coupled with annotations such that the code can be automatically proved to satisfy certain safety conditions. A well-formed TAL program is memory safe (i.e. no pointer forging), control-flow safe (i.e. no jumping to arbitrary memory locations), and stack-safe (i.e. no modifying of non-local stack frames) among other desirable properties. TAL has been implemented for the Intel IA32 instruction set; this implementation, called TALx86 [25], includes a TAL verifier and a prototype compiler from a type-safe C-like language called Popcorn, to TAL. MANE is actually written in Popcorn.

Enhancing a node with a plug-in or update extension is achieved through type-safe dynamic linking [13]. A frequent use of plug-in extensions is to extend the services available to PLAN packets. To do this, extensions are loaded and plugged into the service symbol table. When future APs are processed, they will reference this table, and thus have access to the new functionality. Update

extensions are dynamically linked as well, but differ from plug-in extensions in that the existing node code and any existing extensions are *relinked* following the update [11]. In this way, existing code 'notices' that a new version of a particular module has been loaded the next time that code is called (and the old code will run until this point). Care must be taken when using this technology to avoid errors arising from the timing of an update [11].

MANE presents a two-level namespace architecture. References in the packet to services are resolved by the service plug-in namespace, while references between plug-ins and/or the rest of the program are resolved by the program namespace. In both cases, these namespaces may be changed at runtime to refer to new entities. A benefit of this separation is that the presentation of each namespace can be parameterized by policy, for example to include security criteria. This is useful because the division between the Active Packet, plug-in extension, and the update extension layers constitutes a likely division of privilege. APs are quite limited in what they can do, so we allow arbitrary users to execute those packets. However, when a packet calls a service, implemented as a plug-in extension, the privilege of the packet can be checked before allowing the call to take place [9]. Similarly, when an update extension is loaded, the privilege of those extensions that would relink against the update extension can be checked before allowing the relinking to take place.

The basic network model of MANE is much like an IP network. MANE addresses are globally unique and hierarchical. The hierarchy is based on sub-nets of nodes and individual nodes on a sub-net can broadcast to each other, while communication with nodes on other networks must be mediated by routers. MANE routers run a conventional link-state routing protocol. MANE supports a form of DHCP, which can dynamically assign both an address and a default router to a node connected to a given network. MANE uses an ARP-style protocol to resolve the link-level address corresponding to a network level address and there is a provision for proxy-ARP as well.

There are also key differences between an IP network and MANE. MANE communicates using only APs and nodes can be extended and updated. To support APs, MANE provides certain basic services, such as means to identify a node and to store and retrieve *soft state* based on a key. A soft-store is an essential service for APs and is provided by many AN systems (e.g. [36, 22, 24, 10, 28]).

At its lowest level, MANE emulates broadcast networks by keeping track of which nodes are on a particular sub-net and using UDP to communicate between neighbors. Broadcast is achieved by repeated unicast. This level also supports emulation of physical node mobility, allowing a node to leave a sub-net and to join new sub-nets. To the higher-level software, this emulation is transparent.

3.2 Mobility and Mobility Protocols

Several reasons motivated our choice of mobility from which to draw our examples. First, mobility is an area in which new protocols and improvements to existing protocols are being developed rapidly. Thus it is an area where better evolutionary capabilities could be a real benefit, since then protocols could be

deployed and later upgraded and replaced as new techniques develop. Second, in the particular area of mobile-IP, current protocols are constrained in their design to require only local changes to the network infrastructure. Practical evolution capability would allow other (preferable) protocols to be developed. Finally, mobility is an interesting domain in its own right, and the current work allows us to begin to understand the issues there in the context of our design and implementation environment.

The goal of the work here is not to innovate in the area of mobility. By choice, we have taken existing, well understood mobile protocols and shown how networks can be evolved to support them. We have done this both to gain an understanding of these protocols, but more importantly to avoid clouding the evolution issues with issues involving new mobile protocols. We make no claims that the protocols we use here are the best possible ones, but rather they are known solutions that serve to illustrate how wide classes of protocols can be easily evolved and integrated using AN techniques. For broader considerations about how AN could be applied to mobile networks, see [34, 31].

Mobile-IP To demonstrate both Active Packet and update extension evolution, we have chosen to add support to MANE for what is essentially mobile-IP [29, 30]. Only end nodes can be mobile. A mobile host has a "home" network, which is implicit in its address. Even when a host is mobile, packets for it are sent to its home network for delivery. If a host is not mobile, packets are delivered in the normal way. If a host is mobile, when it connects to a remote or "foreign" network, it uses DHCP to acquire a local address, allowing the node to function as its own foreign agent. The mobile host then sends a registration packet to one of the routers on its home network with the information that it can be contacted at its newly acquired address. This router acts as the "home agent". When a packet arrives at the home agent destined for the mobile host, it is tunneled to the mobile host using its address on the foreign network. There the packet can be removed from the tunnel and delivered.

Consider what must be added to MANE to support this protocol:

1. The home agent must be identified.
2. There must be a way to send a registration packet.
3. There must be a way to recognize when a packet arrives at the home agent.
4. There must be a way to create a tunnel.
5. There must be a way to remove the original packet from the tunnel.

The application-aware and transparent evolutions will share the same implementation for many of these functions. The shared implementation is the inherently non-transparent part of mobile-IP, including basically all but point 3.

Multi-hop Ad-Hoc Routing To demonstrate plug-in extension evolution, we have chosen to show how a multi-hop routing protocol can be added to an ad-hoc mobile network with only neighbor-to-neighbor communication. The protocol we have chosen is Dynamic Source Routing (DSR) [19, 18]. While a variety of

protocols could also be deployed using our techniques, DSR has the advantage of being simple and well understood.

DSR is an on-demand routing protocol, which searches for a source route to some destination by flooding a ROUTE REQUEST packet on the network when the initiating node has data packets to send. This ROUTE REQUEST packet is forwarded on the whole network to the target, which replies with a ROUTE REPLY packet. When the ROUTE REPLY packet arrives, the initiator can start sending data packets using the source route found in the ROUTE REPLY packet. It also saves the route in its route cache. An important optimization is for intermediate nodes to also maintain a route cache so that if they already know a route to the requested destination, they can simply send a ROUTE REPLY, thus limiting flooding. Data in the route caches times out, to avoid stale routes.

Consider what needs to be added to MANE to support this protocol:

1. ROUTE REQUEST packets must be flooded. This includes the need to check if an intermediate node has already seen the particular flood message.
2. There needs to be a route cache that can be queried and updated.
3. There needs to be a ROUTE REPLY packet that carries the source route back to the initiator.

Note that many of the preexisting capabilities of MANE are not needed by DSR, mostly just the ability to send packets one hop and to name nodes.

4 Active Packet Evolution

If it is acceptable for the node trying to communicate with the mobile host to be aware that the host might be mobile, then it is possible to implement the basic mobile-IP protocol discussed in Section 3 using only APs. Some of this implementation can be shared with the update extension evolution example (in Section 6), we discuss this first, followed by a discussion of the aspects unique to APs.

4.1 Setting the Forwarding Path

The application-aware and transparent versions share the same infrastructure for setting up the forwarding path to a mobile host (while they differ in how packets are actually forwarded). Before a mobile node leaves its home network, it must identify the router that serves as its home agent. For simplicity, we assume that its default router serves this purpose. Once the node has attached itself to a new network and has a unique address, it sends an AP containing a control program to register itself to its home agent. When executed, this program simply adds the information to the home agents soft-state keyed by the mobile nodes home network address. Both the application aware and transparent versions share the same soft-state entries, allowing them to use the same control program and to co-exist.

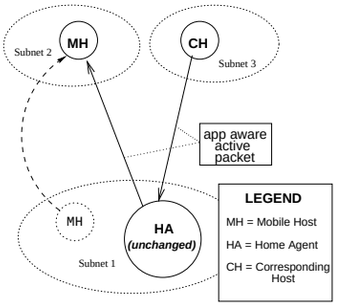

Fig. 1. Active Packets for Mobile-IP

4.2 Forwarding: The Application-aware Protocol

The key questions remaining are how do we detect that a packet is at the home agent of a mobile host and how is the packet then tunneled to the unique address. Because this version is application aware, both of these steps can be done by having the application use a special AP as shown in Figure 1.

The PLAN code for the packet that must be sent by the application is shown in Figure 2. `GetToAgent` is the main function and when it executes, it first looks up `dest` in the soft-store using `lookupTuple`. If that succeeds, it has found the home agent and it uses `OnRemote` to send a new packet, the tunnel, that will execute `FoundFA` at the foreign agent with the same arguments as `getToAgent`. `OnRemote` provides multi-hop transmission of the packet with out execution until it reaches the foreign agent. If the lookup fails the handle will execute. If we have actually reached the host then we deliver the packet. Otherwise, it looks up the next hop toward `dest`. It then uses `OnNeighbor`, which only transmits a packet one hop, to send the packet. Thus the packet travels hop-by-hop looking for the home agent.

Now consider the `FoundFA` function. It executes on the foreign agent, which is in our case is the mobile host, but might be some other node on the same sub-net. It sends a packet to the `dest` that does the deliver. This is where the original packet is removed from the tunnel. Notice that all of that functionality is encoded in the tunnel packet program itself, the foreign agent does not need to have any knowledge of its role as a tunnel endpoint, it just has to support PLAN.

5 Plug-In Extension Evolution

Some evolutions do not just augment or customize some existing aspect of the network, but rather add wholly new functionality. Such evolutions may be problematic with just APs because of their 'fixed vocabulary.' To rectify this problem, we can load new services in the form of plug-in extensions to extend a packet's

```
fun getToAgent(dest, payload, port) =
 try
  let val fagent = lookupTuple(dest) in
   OnRemote(|FoundFA|(dest, payload, port), fagent, getRB(), defRoute)
  end
 handle NotFound =>
  if (thisHostIs(dest)) then deliver(payload, port)
  else
   let val next = defaultRoute(dest) in
    OnNeighbor(|getToAgent|(dest, payload, port), #1 next, getRB(),
    #2 next))
   end

fun FoundFA(dest, payload, port) =
 let val hop = defaultRoute(dest) in
  OnNeighbor(|deliver|(payload, port), #1 hop, getRB(), #2 hop))
 end
```

Fig. 2. PLAN packet for Mobile-IP

Algorithm 5.1: ROUTE DISCOVER(*Simple DSR*)

procedure ROUTE REQUEST(*Target*, *RouteRec*)
 if *Duplicate Request Packet Or My Address Already In Route Rec*
  then *Discard*
  else **if** *This Host Is Target*
   then ROUTE REPLY(*RouteRec*)
   else *Append My Address To Route*
    Flood To All Neighbors

vocabulary, and thus enable implementation of the new networking functionality. More generally, if a node defines an extensible interface, new extensions can be loaded and plugged into this interface to provide extended or enhanced functionality. This is the essence of plug-in extension evolution.

As we will show, it is possible to add multi-hop routing based on DSR to an ad-hoc network only supporting neighbor communication by using plug-in extension evolution. The main functionality is embodied in a special AP for communicating routes between nodes, which makes use of a non-standard service. This packet does not need to be sent by an application, but rather by the networking software on the ad-hoc node, and therefore the dynamic deployment of the protocol does not require applications to be aware of the change.

5.1 An Active Packet for Route Discovery

Consider the pseudocode shown in Algorithm 5.1 for a simple AP that implements route discovery. The packet itself embodies many key aspects of the pro-

tocol directly. In particular, it does duplicate elimination, tests for routing loops, detects termination and sends a reply, and performs flooding. In general, since many protocols have relatively simple control flow, simple APs can implement key aspects of the protocol directly. However, this algorithm benefits from node-resident services that are specific to the protocol, particularly to detect if this is a duplicate request or if the current node is already in the route record.

5.2 Deploying DSR Dynamically

Implementing the required node-resident services is straightforward. To deploy the service dynamically, we need a way to discover the nodes that will need the service, upload the extension to those nodes, and install it. Since a general treatment of node-resident service discovery and deployment is a topic of research, consider the following basic idea. One way to discover which nodes will need an extension is to require that when a packet arrives at a node, it ensures that extensions it needs are installed before it executes. Acquiring a needed extension could be done in a number of ways. For example, it could always carry the extension with it, it could retrieve the extension from a well-known code repository, or it could use an ANTS-style distribution protocol [36] in which the extension is obtained from the source of the current packet. For our demonstration we chose the simplest option—carrying the extension in the route request packet—but any of these solutions could be implemented in MANE. The key point is that given the ability to dynamically load code, the network itself provides the mechanisms to transport loadable code to the locations where it is needed. The pseudocode for the simple approach is shown in Algorithm 5.2.

Algorithm 5.2: ROUTE DISCOVER($SimpleDSR + DynamicLoading$)

procedure ROUTE REQUEST($Target, RouteRec, Extension$)
 if *DSR Service Not Present*
  then *Load DSR Extension From This Packet*
 Same Code as Above

6 Update Extension Evolution

A potential problem with the evolutions shown so far is that some part of the system must send special packets to take advantage of the new service. In some applications, being aware of new packets or services is acceptable, while in others it is not. The latter is true for mobile IP: it would be unreasonable to change all possible senders on the network to use our special packets from Section 4 to send to a potentially mobile host. In this section, we demonstrate how using update extensions, we are able to evolve the network so that forwarding is transparent to the sender and does not require using a special packet.

The basic strategy is shown Figure 3. Here a packet that is not aware of mobility is destined for a mobile node. However, because we have updated the Home Agent to support transparent forwarding, it is able to intercept the packet and tunnel it to the mobile host. Thus, although we use Active Packets and

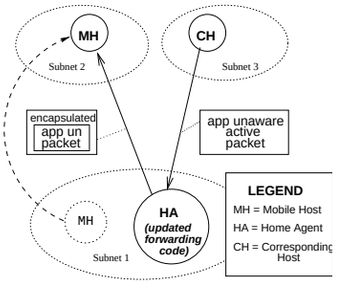

Fig. 3. Update extension Mobile-Ip evolution

plug-in extensions to help perform our evolution in a convenient way, the key to transparent evolution is really the use of update extensions.

Because mobility is inherently not transparent to the mobile host itself, we can reasonably have it set up the forwarding path to the remote agent as described in Section 4.1, with the added benefit that the nontransparent and transparent techniques can coexist.

6.1 Forwarding: The Application-Transparent Protocol

To make the forwarding transparent to the sender requires a way to detect that a packet has arrived at the home agent and to forward it to the foreign agent *without having to rewrite the sender's packet code*. The most straightforward way of doing this is to modify the router's forwarding logic: whenever a packet arrives, look up its destination address in the soft state that is used to record which hosts are mobile, and if present, forward the packet to the foreign agent. Pseudocode for router forwarding logic is shown below (in a C-like syntax), with the additional part shown in italics:

```
void sendToNextHop(pkt_t packet, host_t dest){
  host_t nextHop = Route(dest);
  if is_mobile_host(dest) then
    tunnel_to_foreign_agent(packet, dest);
  else
    send_on_link(packet, dest, next_hop);
}
```

Note that the code implementing `is_mobile_host` and `tunnel_to_foreign_agent` would be implemented elsewhere and loaded separately.

While the addition of an if-statement to the forwarding loop is conceptually simple, it is impossible to realize *on-the-fly* without the support of update extensions. This is because the forwarding loop in MANE was not designed for change; that is, it did not provide a plug-in interface for performing new operations in

the loop.[1] As a result, effecting this change would require changing the code statically and recompiling, and then bringing down the node and restarting it with the new code. This is practical when only a few nodes need to be updated, but much less so if an evolution needs to be widespread.

On the other hand, the power of update extensions makes them more dangerous. For example, the added conditional in the forwarding loop above will be invoked for *all* packets, even those not interested in mobility. In an active packet-only system, the needs of one packet will not interfere with another's in this way. Similarly, allowing multiple users update arbitrary parts of the router's code could result in incompatible changes, and/or an unintelligible code base. As such, update extensions will likely be limited to privileged users, limiting their applicability.

Implementing this change as an update extension in MANE requires two actions. First, dynamically load and link a mobility module that implements the test of whether a packet needs to be forwarded as well as providing the tunneling code. Second, dynamically load a new version of the forwarding code that does the required test and then update the old running code with the new version.

One interesting point remains. How is the tunnel itself created? In essentially the same way as in the non-transparent case, although obviously done by the node resident mobility code. A new AP is created which when executed on the foreign agent unpacks the original packet and delivers it. Note that tunneling this way works even when the packet being tunneled is not active and thus again avoids the need for the foreign agent to act explicitly as the tunnel end-point.

7 Conclusions

To conclude, we summarize the lessons learned while demonstrating each of the three kinds of evolution.

Active Packet Evolution The chief advantage of Active Packet evolution is that it is lightweight and allows third parties to enhance the functionality of the network without changing the nodes themselves. Thus, from the point of view of security, this style of evolution is the most desirable and gives the widest variety of users the ability to evolve the network. One disadvantage is that it is inherently not application transparent. Another key disadvantage is that if the existing node interface does not support some critical piece of functionality, it may be impossible to achieve the desired result. Despite this, our example (and others, e.g. [22, 21]) shows that even with only very basic services, non-trivial applications are feasible. An interesting challenge to the AN community is to design a set of node-resident services that maximizes the range of evolutions that can be achieved with just APs. Since, as in our example, the APs can

[1] We could imagine designing the forwarding loop to allow for extensibility, as is the case in CANES [3]. However, there will always be parts of the system that were not coded to anticipate future change, and therefore will lack a plug-in interface. As a result, these parts of the system can only be updated if with update extensions.

often embody a substantial part of the control aspect of a protocol, this effort would be quite different from typical protocol design and would need to focus on providing the generic components that support the aspects of a variety of protocols that can not be expressed in the packets. Interestingly, just the simple soft state provided by ANTS [36] and our system, is already a significant step in that direction.

Plug-in Evolution Plug-in extensions provide significantly greater evolution possibilities when used with APs. This is because protocols can have both new packet programs and new node resident components. This power comes directly from having the service namespace in the AP system provide a generic plug-in interface. One obvious disadvantage is that such service plug-ins are not application transparent.

Our examples did not take advantage of plug-in extensions that were not accessed from packet programs, as is possible in CANES [3], and Netscript [37], among others. Such plug-ins have the advantage that they can be used to achieve application-transparent evolution, but only in ways that the underlaying system has anticipated by providing a plug-in interface.

A general disadvantage of plug-in evolution is that dynamically loading plug-ins implemented using general purpose programming language code poses a significant security risk. This will restrict the range of users who can deploy these kinds of extensions. This suggests another research opportunity for the AN community, designing plug-in extensions that can be safely loaded by third parties. One approach to this would be to use special-purpose languages with built-in security properties, in the spirit of PLAN.

Update Evolution The example here is the first example of update evolution in AN, chiefly because MANE is the first AN system to support such evolution. The advantage of this approach is clear: application-transparent evolution can be achieved even when the system design has not anticipated the need for a particular kind of change. In some sense, this embodies the entire goal of AN. There are two disadvantages. One, dynamic updating is not a widespread technology like dynamic loading, though it can be conceptually simple to implement [11]. Second, and more importantly, the power of update extensions implies the need for greater security and reliability considerations than for plug-in extensions or active packets. In the short term, this means that only privileged users with access to the entire router code base should make use of this technology. In the long term, more research is needed to understand how to manage multiple updaters of the same code, and ways to limit their system-wide effects.

We have presented a taxonomy of types of AN evolution driven by the underlaying language technology. Based on that taxonomy, we presented a series of examples that illustrate what expressibility gains are possible as successively more powerful techniques are used. However, these gains in expressibility are balanced by the greater security risks of more powerful techniques. Greater security risks imply that fewer users may deploy a system. Thus a basic design principle for AN systems should be to use the least powerful evolutionary mechanisms possible so as to maximize the range of users that may deploy a system.

References

[1] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. A Secure Active Network Environment Architecture: Realization in SwitchWare. *IEEE Network Magazine*, 12(3):37–45, 1998. Special issue on Active and Controllable Networks.

[2] D. S. Alexander, M. Shaw, S. M. Nettles, and J. M. Smith. Active Bridging. In *Proceedings, 1997 SIGCOMM Conference*. ACM, 1997.

[3] S. Bhattacharjee. *Active Networking: Architecture, Compositions, and Applications*. PhD thesis, Georgia Institute of Technology, August 1999.

[4] K. Calvert, S. Bhatacharjee, E. Zegura, and J. P. Sterbenz. Directions in Active Networks, October 1998.

[5] D. Decasper, Z. Dittia, G. M. Parulkar, and B. Plattner. Router plugins: A software architecture for next generation routers. In *SIGCOMM*, pages 229–240, 1998.

[6] D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf, and B. Plattner. A scalable, high performance active network node, 1999.

[7] O. Frieder and M. E. Segal. On Dynamically Updating a Computer Program: From Concept to Prototype. *Journal of Systems and Software*, 14(2):111–128, September 1991.

[8] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, pages 86–93. ACM, 1998.

[9] M. Hicks, A. D. Keromytis, and J. M. Smith. A Secure PLAN (Extended Version). In *Proceedings of the DARPA Active Networks Conference and Exposition (DANCE)*. IEEE, May 2002.

[10] M. Hicks, J. T. Moore, D. S. Alexander, C. A. Gunter, and S. Nettles. PLANet: An Active Internetwork. In *Proceedings of the Eighteenth IEEE Computer and Communication Society INFOCOM Conference*, March 1999.

[11] M. Hicks, J. T. Moore, and S. Nettles. Dynamic Software Updating. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 13–23. ACM, June 2001.

[12] M. Hicks and S. Nettles. Active Networking means Evolution (or Enhanced Extensibility Required). In *Proceedings of the Second International Working Conference on Active Networks*, October 2000.

[13] M. Hicks, S. Weirich, and K. Crary. Safe and Flexible Dynamic Linking of Native Code. In *Preliminary Proceedings of the ACM SIGPLAN Workshop on Types in Compilation*, Technical Report CMU-CS-00-161. Carnegie Mellon University, September 2000.

[14] K. Hino, T. Egawa, and Y. Kiriha. Open programmable layer-3 networking. In *Proceedings of the Sixth IFIP Conference on Intelligence in Networks (SmartNet 2000)*, September 2000.

[15] G. Hjálmtýsson and R. Gray. Dynamic C++ Classes, A lightweight mechanism to update code in a running program. In *Proceedings of the USENIX Annual Technical Conference*, June 1998.

[16] L. Hornof. Self-Specializing Mobile Code for Adaptive Network Services. In *Proceedings of the Second International Working Conference on Active Networks*, volume 1942 of *Lecture Notes in Computer Science*. Springer, 2000.

[17] A. W. Jackson, J. P. Sterbenz, M. N. Condell, and R. R. Hain. Active Monitoring and Control: The SENCOMM Architecture and Implementation. In *Proceedings of the DARPA Active Networks Conference and Exposition (DANCE)*, May 2002.

[18] D. Johnson, D. Maltz, and J. Broch. DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In C. E. Perkins, editor, *Ad Hoc Networking*. Addison-Wesley, 2001.

[19] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[20] S. Karlin and L. Peterson. VERA: an extensible router architecture. *Computer Networks (Amsterdam, Netherlands: 1999)*, 38(3):277–293, 2002.

[21] U. Legedza, D. Wetherall, and J. Guttag. Improving the Performance of Distributed Applications Using Active Networks. In *IEEE INFOCOM*, March 1998.

[22] L. Lehman, S. Garland, and D. Tennenhouse. Active Reliable Multicast. In *IEEE INFOCOM*, March 1998.

[23] S. Malabarba, R. Pandey, J. Gragg, E. Barr, and J. F. Barnes. Runtime support for type-safe dynamic Java classes. In *Proceedings of the Fourteenth European Conference on Object-Oriented Programming*, June 2000.

[24] J. T. Moore, M. Hicks, and S. Nettles. Practical programmable packets. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, April 2001.

[25] G. Morrisett, K. Crary, N. Glew, D. Grossman, R. Samuels, F. Smith, D. Walker, S. Weirich, and S. Zdancewic. TALx86: A Realistic Typed Assembly Language. In *Second Workshop on Compiler Support for System Software*, Atlanta, May 1999.

[26] G. Morrisett, D. Walker, K. Crary, and N. Glew. From System F to Typed Assembly Language. *ACM Transactions on Programming Languages and Systems*, 21(3):527–568, May 1999.

[27] G. Necula. Proof-Carrying Code. In *Twenty-Fourth ACM Symposium on Principles of Programming Languages*, pages 106–119, Paris, Jan. 1997.

[28] E. Nygren, S. Garland, and M. F. Kaashoek. PAN: A high-performance active network node supporting multiple mobile code systems. In *OPENARCH'99*, March 1999.

[29] C. Perkins. IP mobility support. Internet RFC 2002, October 1996.

[30] C. Perkins. IP mobility Support Version 2. Internet Draft, Internet Engineering Task Force, Work in progress., 1997.

[31] B. Plattner and J. P. Sterbenz. Mobile wireless activenetworking: Issues and research agenda. In *IEICE Workshop on Active Network Technology and Applications (ANTA) 2002*, Tokyo, March 2002.

[32] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. Smart packets: Applying active networks to network management. *ACM Transactions on Computer Systems*, 18(1), February 2000.

[33] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[34] C. Tschudin, H. Lundgren, and H. Gulbrandsen. Active Routing for Ad Hoc Networks, April 2000.

[35] I. Wakeman, A. Jeffrey, T. Owen, and D. Pepper. SafetyNet: A language-based approach to programmable networks. In *OPENARCH'00*, April 2000.

[36] D. J. Wetherall, J. Guttag, and D. L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH*, April 1998.

[37] Y. Yemini and S. daSilva. Towards programmable networks, 1996.