An Access Control Architecture for Programmable Routers

Jun Gao[1] Peter Steenkiste[1,2]
[1]School of Computer Science
[2]Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{jungao, prs}@cs.cmu.edu

Abstract— **Programmable networks allow the router's functionality to be extended dynamically through the use of active extensions. This flexible architecture facilitates the deployment of new network protocols and services. However, the programmable nature of a network also raises serious safety and security concerns. These concerns must be addressed before programmable networks can be deployed. One particular security question is how we can limit what resources and data active extensions can access on the router. While existing operating systems address this question for end-points and servers, routers have been designed to perform a different task, namely forwarding packets, and the existing OS solutions turn out to be inadequate for routers. In this paper we look at how we can restrict active extensions' access to link bandwidth and data traffic. Our solution is based on access control lists that are used to check all active extensions' operations that may affect the use of link bandwidth, or may involve access to user traffic. We implemented these mechanisms in Darwin, an example of a programmable network.**

Keywords— **Programmable networks, Active extensions, Security, Access control**

I. Introduction

Traditional network devices, such as routers in the Internet are closed systems that implement a fixed set of functions. Software that runs on a router is typically supplied by the router vendor and the customer's control over routers is limited to managing built-in functions. This type of router design slows down the deployment of new services since all changes or extensions to the router functionality have to be implemented by the vendors. An open router architecture that can execute software from a wide range of sources potentially allows much more rapid innovation. For example, third-party software vendors can implement diverse network Quality of Service (QoS) packages for service providers with different requirements. Similarly, programmable routers allow the deployment of VPN service that supports customized per-VPN QoS and network management [15]. We will call software modules that extend router functionality "extensions".

While a programmable router architecture increases flexibility, it also raises safety and security concerns. Safety centers around the question of how we can safely execute extensions that may be faulty, e.g., they could cause the router or router components to fail. These problems are addressed by isolating the extension code from the rest of system, either using runtime mechanisms (e.g., Java sandboxing, virtual memory) or compile time mechanisms (e.g., Proof carrying code [17]). In this paper we focus on security issues, i.e., we want to prevent router extensions from disrupting the network service received by other users, for example by using their resources or by reading or writing their data. This problem is similar to the problem addressed by a traditional operating system, except that routers have a very different task. Their primary responsibility is forwarding and processing packets, not general-purpose data processing, storage management, or user interface support. This means that router operating systems face a different set of security concerns.

In order to perform their tasks (e.g., implementing QoS, selecting routes, or encrypting data), router extensions must be able to control critical router resources such as link bandwidth and access critical data structures such as the routing table. Extensions must also be able to manipulate data traffic, e.g., dropping packets or modifying packet contents. It is easy to see that without proper security mechanisms, extensions can use these operations to harm other users. For example, malicious or faulty extensions can "steal" bandwidth by making invalid reservations, can corrupt the routing table, or can manipulate data traffic that belongs to other users. In this paper, we present a set of security mechanisms for programmable networks that can prevent router extensions from affecting the performance or traffic of other users. We focus on dealing with threats that are router specific, e.g., unauthorized use of link bandwidth or access to traffic. We will not deal with more general challenges such as protecting memory or CPU cycles, since these problems can be addressed using traditional operating system techniques.

The rest of the paper is organized as follows. In Section II, we motivate the security problems by examining the requirements of a programmable network and the basic operations within such a network. In Section III, we present the design of our security system. We elaborate on the access control mechanism on routers in Section IV and we discuss the policy-based active extensions security

This research was sponsored in part by the Defense Advanced Research Project Agency and monitored by AFRL/IFGA, Rome NY 13441-4505, under contract F30602-99-1-0518.

control in Section V. We describe our implementation in Section VI. We present related work in Section VII and we conclude in Section VIII.

II. Motivation

The functions of a typical router can be divided into two planes, a control plane and a data plane. The data plane is responsible for packet forwarding, while the control plane supports protocols for routing, signaling, network management, etc. In a programmable router it may be possible to extend the functionality of the control plane, the data plane, or both. It is easiest to add extensions to the control plane since control plane operations are less performance critical. Control plane extensions can then modify or monitor the packet processing in the data plane indirectly through an interface that separates the two planes. In contrast, router functionality can be extended more directly using data plane extensions. We distinguish two types of data plane extensions. The first type of extensions manipulates the data portion of the packets, e.g., video transcoding, compression and encryption; these operations typically require significant computing capabilities in the data plane. The second type can change how packets are forwarded or what kind of quality of service packets may receive. These operations do not require payload processing, so they are less expensive. While router functionality can be extended on the fly using active packets or capsules [22], we assume in this paper that active extensions are explicitly installed using a signaling protocol.

An example of the above programmable router architecture is the CMU Darwin system [5]. Control plane extensions in Darwin are called *delegates* and they are installed using the Beagle signaling protocol [6], [7]. Figure 1 depicts the basic Darwin router architecture. Delegates execute in the control plane and they can affect the forwarding behavior of the router for specific flows by making calls to the Router Control Interface (RCI) [21], [10]. Using the RCI, delegate can control data plane components such as packet classifiers, schedulers, monitoring modules, and route lookup. Besides these standard components, the Darwin data plane also has packet processing modules (PPM). PPMs can perform a wide range of operations on data flows, e.g., selective packet dropping, flow redirection, tunneling, or payload data processing. PPMs are pre-installed and delegates can specify what processing should be applied to specific flows. In a more flexible implementation, additional modules could also be installed on the fly, i.e., the packet processing modules can be thought of as simple, pre-installed data plane extensions [1].

The RCI consists of a set of methods that supports a rich set of operations on flows, where a flow is defined as a sequence of packets that belong together, as defined by a *flow spec* [24], [5]. RCI methods fall in three categories [11], [10]. The first category of methods enables delegates to manipulate flows by updating the input or output classifier data structures. For example, a delegate can identify a new flow by providing a flow spec that characterizes the packets in the flow. The delegate can then apply further processing to this flow.

[1]Darwin is implemented in FreeBSD and NetBSD-based routers that do not easily support on-the-fly data plane extensibility.

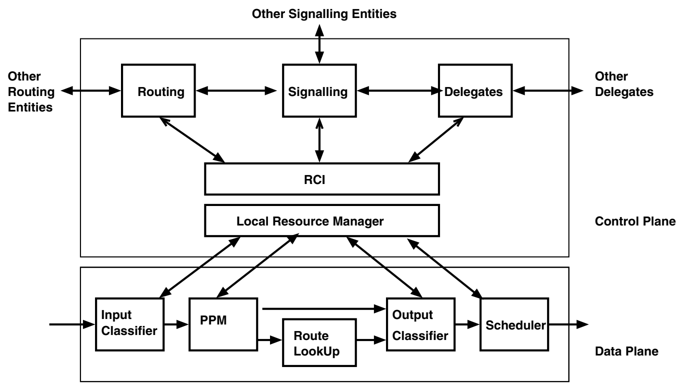

Fig. 1. Darwin node architecture.

The second category of methods deals with the quality of service flows receive, i.e., it controls packet scheduling. Darwin uses a hierarchical scheduler [20], which means that the bandwidth distribution across flows is controlled using a resource tree. The RCI methods allow delegates to modify the resource trees for the output links (e.g., make, modify, or terminate bandwidth reservations) and to associate flows with nodes in the tree to provide bandwidth guarantees to a class of traffic.

The third category of RCI methods deals with more general packet processing or monitoring operations, i.e., delegates can specify what processing should be applied to specific flows by selecting a PPM and specifying appropriate parameters the PPM may require.

The primitive methods of the RCI provide the building blocks for delegates to build fairly sophisticated services. For example, delegates can implement specialized routing protocols or a customized network monitoring infrastructure. However, given their ability to access critical router resources and affect data flows, delegates can pose serious threats to the programmable network and other users. The threats range from risks local to one router to risks that can span the whole network. For example, without proper access control, a delegate can steal bandwidth for its flows by either increasing its bandwidth reservation parameters, or by associating its flows with resource nodes of other users. Alternatively, a malicious delegate can reroute random flows to disrupt other users' traffic, or issue a Denial-of-Service (DoS) attack by tunneling flows to a victim server or network segment.

Besides managing link bandwidth and controlling traffic, delegates can also communicate with applications or other delegates using standard, socket-based communication mechanisms. Communication further complicates the way we want to control delegates' access to bandwidth and traffic. For instance, using inter-delegate communication, a delegate may delegate certain traffic or bandwidth related tasks to other delegates. A specific example could be a delegate that is in charge of flows belonging to a video streaming application, and that may want to ask a specialized monitoring delegate to monitor specific subflows.

Even more intriguing scenarios unfold when we allow delegates to create other delegates. A motivating example is a network infrastructure that supports a number of VPNs, where each VPN may want to deploy its own signaling protocol that is different from the "standard" signaling protocol used by the network. As mentioned above, in Darwin-like programmable networks, we rely on the signaling protocol to install extensions (i.e., delegates), which means that if we want to dynamically install new signaling protocols as delegates, these signaling delegates must be able to install other delegates. These advanced operations make it even harder to keep track of what operations delegates are allowed to perform.

III. Delegate Security Design

The security concerns for programmable routers raised in the previous section center around the question of what router resources a delegate should be able to access. There are two types of resources that are unique to routers and are not explicitly dealt with by traditional operating systems: (1) resources on the router, e.g., link bandwidth, and routing tables; and (2) data traffic that is passing through the router. To protect these resources from being mis-used or mis-treated, the programmable router must be equipped with proper access control mechanisms.

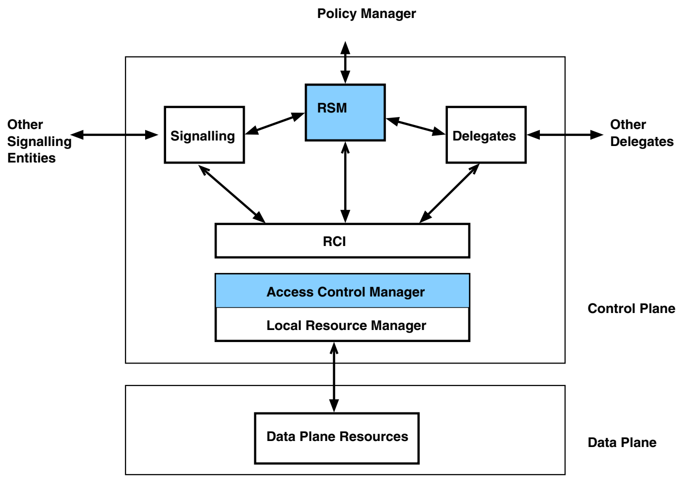

Fig. 2. Secure Node Architecture (some security unrelated components are omitted).

It may seem to be conceptually straightforward to design a security system that can limit access to these resources: we merely need to add an extra layer of permission checking whenever delegates access resources (Figure 2). That is indeed the basic idea behind our security system design and it is a simple and effective way of controlling delegates that utilize primitive RCI methods. For example, consider a monitoring delegate periodically examines the bandwidth consumption of a particular flow. All the router needs to do is to check whether the delegate has appropriate access permissions for the flow. This approach is not only simple, but also efficient, since the checking can be done locally on the router. This is important, because one of the fundamental reasons that delegates are set up to run in the routers is that they can react to network conditions much faster than the endpoints they represent, and if common operations must require extensive security checks by a remote entity, delegates would become ineffective.

Unfortunately, not all the access control decisions can be made locally and conveniently. Consider the scenario in which we have a signaling delegate that makes bandwidth reservations and creates delegates on behalf of a new user. Clearly the router must decide whether the new user is allowed to further allocate bandwidth and create other delegates. This type of decision requires more extensive policy checking, and it is not practical to have the necessary policy information stored on every router so that it can make a decision locally. Instead, it is more appropriate to rely on an external policy manager, that can be consulted by all the routers in the administrative domain. This design is similar to the policy-based admission control framework developed for IP networks [23].

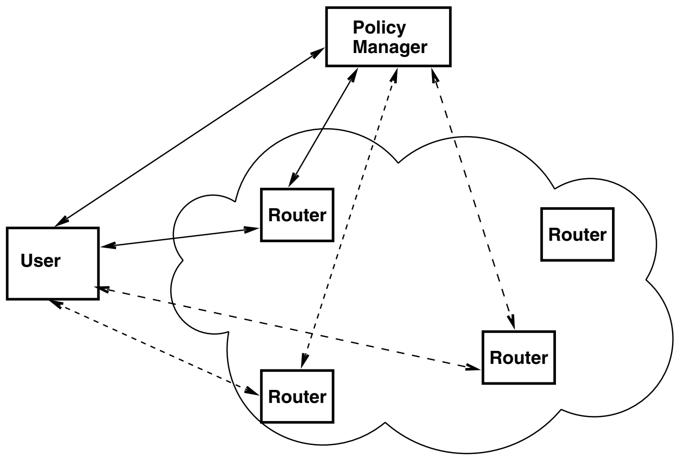

Fig. 3. System architecture.

Figure 3 shows the overall architecture of our security design. The user is the entity that injects delegate code into routers; it is also known as the Delegate Initiator (DI). We designed a protocol for the User, Policy Manager (PM) and Router to securely exchange security policy information and delegate code. Routers enforce access control in one of two ways. Simple, common operations that may affect bandwidth use or may involve access to traffic are checked by the Access Control Manager (ACM) based on a set of access control lists (Figure 2). More complex operations are checked by a centralized PM. The Router Security Manager (RSM) is responsible for executing these operations, after they are approved by the PM.

In the Darwin implementation of this architecture, the ACM is responsible for checking all simple RCI calls, while the RSM is responsible for the creation of delegates, which involves setting up new access control lists. This separation of local access checking and remote policy making provides a good balance between the need for good delegate performance and the desire for router simplicity. Most of the delegate's time-critical operations use the RCI and they should be checked efficiently. More complex operations such as installing delegates are rare and less time critical, so decisions can be made on a slower time scale.

In the next section, we describe in more detail how the ACM protects bandwidth and traffic, while the interactions between the DI, ACM, RSM, and PM are described in Section V.

IV. Access Control Design

As observed earlier, delegates manipulate link bandwidth or access data traffic by invoking primitive RCI methods. This is similar to the scenario in a shared file system, where users access directories and files via standardized file I/O system calls. The analogy inspires the use of access control lists (ACL) to protect the data plane resources. An ACL typically consists of three parts: a principal, an object and a permission string that defines what permission this principal has on this object. The principals in this context are delegates. We use different types of objects to represent the abstract resources to be protected. For example, portions of link bandwidth are represented by nodes in the resource tree, while data traffic is classified into flows. A set of permission bits are designed for each type of objects to define the possible operations on these objects. When a delegate has some permissions on an object, the corresponding bits in the permission string will be turned on.

The key data structure in such a system is an access control matrix, which is essentially a table, where each row represents a principal, each column represents an object, and each entry is the set of access rights for that principal to that object. We use the following format to represent each entry of the matrix:

`[ delegate-id, object-id, rights ].`

In this section, we present the design of two types of ACL and we describe how they are used to protect the router's link bandwidth and user data traffic respectively.

A. Protecting Bandwidth

As we described in Section II, Darwin uses a hierarchical packet scheduler since it allows both the support of link sharing and finer grain bandwidth management. In particular, a hierarchical scheduler represents the division and sharing of bandwidth on a link in the form of a hierarchical resource tree. Each node in the resource tree corresponds to a portion of the bandwidth allocation and the subtree rooted at that node specifies how that bandwidth slice should be further partitioned. Hierarchical scheduling can for example be used to distribute the bandwidth of a link across a set of organizations, where within each organization, bandwidth can be further distributed across departments or applications. The hierarchical fair service curve (HFSC) scheduler [20] used in Darwin has the attractive property that bandwidth allocation decisions made in one subtree of the resource tree do not affect the QoS properties of traffic flows using other subtrees, i.e., there is a good isolation between the subtrees, allowing them to be managed independently.

Delegates perform bandwidth management by manipulating the tree structure through RCI methods. For example, a delegate can reserve bandwidth by adding a new node to the tree via the call **create_node** (*parent_node_id, QoS_parameters*)[2]. It is possible that a malicious delegate can steal bandwidth by adding nodes to a part of the tree that belongs to other users. It can also easily read, change or even remove reservations made by others through other manipulations of the resource tree. Thus, to protect link bandwidth from being abused, a router must protect the resource tree data structure.

Given the properties of hierarchical resource management, it is natural to base the access control for bandwidth on the resource tree. Specifically, we associate delegates with one or more nodes in the resource tree (see Figure 4). The implication is that a delegate is only allowed to make bandwidth management decisions for the bandwidth allocated to that node, i.e., it only controls the properties of that node and its children.

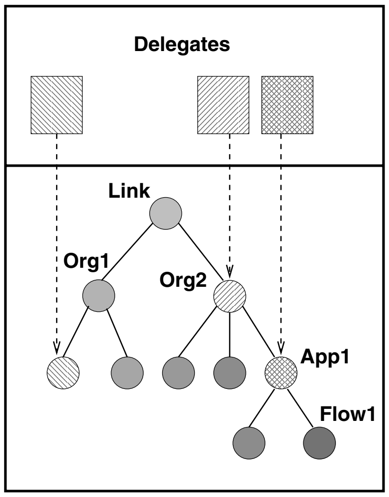

Fig. 4. The association of delegates and resource nodes.

Each node in the resource tree is identified by a unique `node-id` and represents a portion of the bandwidth. By examining the RCI methods related to resource nodes, we construct the following set of access rights to control delegate operations on nodes.

- create (c)

The (c) permission on a node allows a delegate to create new nodes rooted at this node. This means that the delegate can sub-divide the bandwidth that this node represents.

- modify (m)

The (m) permission on a node allows a delegate to modify the bandwidth distribution across its children.

- delete (d)

The (d) permission on a node allows a delegate to delete child nodes.

- retrieve (r)

The (r) permission on a node allows a delegate to retrieve the subtree structure rooted at this node.

[2]For a complete list of RCI methods, refer to [11].

• monitor (n)
The (n) permission on a node allows a delegate to monitor bandwidth usage of the node.
• use (u)
The (u) permission on a node allows delegate to use the resource represented by this node to serve traffic flows for QoS purposes. How specific flows can be served is described later in the paper.

When a delegate is assigned (c) right on a node, (m), (d) and (r) rights must also be assigned to it so that the delegate can modify the bandwidth allocation to children nodes. There are cases that a delegate may only own (m) or (d) rights but not (c) rights on a node. For example, a delegate may be allowed to redistribute bandwidth among a node's children, but it cannot add new children nodes to the node. By maintaining these rights separately, the access control mechanism is more flexible.

TABLE I
Example Access Control List for control of link bandwidth

delegate-id	node-id	access rights
11	3	`cmdrnu`
11	5	`----n-`
12	4	`-mdrn-`
13	5	`----nu`

Table I shows four example ACLs. Delegate 11 has access rights on two nodes (3 and 5). It has the full control over resource node 3: it can create/delete child nodes, modify the resource distribution across the child nodes, monitor the bandwidth usage of the node, and it can also use the bandwidth to serve flows. On the other hand, Delegate 11 can only monitor the bandwidth usage of node 5. Delegate 12 can change the bandwidth distribution among the children of node 4 and can monitor the bandwidth usage of node 4, but it cannot use node 4's resource to serve other flows. Delegate 13 can monitor the bandwidth usage of node 5 and can use node 5's bandwidth to serve flows.

B. Protecting Traffic

In a flow-based programmable network like Darwin, as mentioned in Section II, traffic is classified into flows based on the filters installed in the classifiers of the router. Two types of services can be provided to flows: (1) QoS provisioning, e.g., bandwidth guarantees can be provided to flows by using reserved resources; and (2) special traffic treatment such as flow redirection and data transcoding. By using the RCI primitives, delegates in Darwin can add filters to identify flows, can assign flows to resource nodes to receive bandwidth guarantees, and can associate flows with packet processing modules for customized traffic treatment.

However, without proper control over which traffic can be accessed by which delegates, a malicious or faulty delegate can instruct the router to process traffic belonging to other users, or it can arbitrarily redirect traffic to disrupt the network. To prevent such security violations, a router must (1) constrain what kind of filters a delegate can install to control what traffic it can access, and (2) control the services that can be applied to the traffic that a delegate is responsible for. We first introduce the concept of a filter envelop and then explain the access control mechanisms that govern the processing of traffic.

B.1 Definition of Filter Envelop

Delegates access data traffic by installing filters into classifiers. We associate with each delegate a "filter envelop", that defines the set of filters that can be installed by a delegate on a router and what operations the delegate can perform on the traffic covered by each filter.

A traffic envelop consists of one or multiple *envelop-spec* entries. Each envelop-spec entry has the following format:
`[ traffic-spec, access-rights]`,
where traffic-spec specifies a class of filters that can be defined, and access-rights lists the operating permissions for the traffic defined by those filters. We elaborate on the definition of the traffic-spec in this section, while the access rights are discussed in the next section.

The traffic-spec consists of the following fields: Source IP address, Source IP address mask, Destination IP address, Destination IP address mask, Source port, Destination port, Protocol ID, and Application ID (IP option). The IP address and mask together define the range of possible IP addresses that can be used by a filter. For example: `[ 128.2.205.111, 255.255.255.255 ]` defines a specific IP end-host address and `[ 128.2.0.0, 255.255.0.0 ]` defines an IP subnet address. The source/destination port numbers can be specified as a set of discrete numbers, e.g., `[ 7000, 9000 ]`, or it can be specified as a range of numbers, e.g., `[ 1 - 1024 ]`, or it can be a mix of these two formats: `[ 80, 8080 - 8800 ]`. Both the protocol ID and Application ID fields are specified as a set of numbers, and the set may contain one or multiple numbers, e.g., `[ 6, 17 ]`. For any field, a value of 0 means "Don't Care".

When a delegate creates a new filter, the router checks whether this delegate is allowed to install such a filter based on the envelop that is associated with the delegate. A filter is defined by a *filter-spec*, which consists of the same set of fields as in a traffic-spec, except that the port numbers, protocol ID and application ID can only be specified as a single number instead of a set of numbers. A delegate can only create a filter if the filter-spec is "within" one of the traffic-spec entries of the delegate's filter envelop. By "within" we mean:
• The source/destination IP address of the filter-spec must match the corresponding address specified in the traffic-spec using the longest prefix matching algorithm, e.g., [128.2.205.111 255.255.255.255] matches [128.2.0.0 255.255.0.0].
• The port numbers must be within the set of allowable ports, e.g., port 8080 is in the set of [80, 8080].
• The same rules apply to protocol ID and application ID.
• For a field specified with 0 in the traffic-spec, any value of the same field in the filter-spec matches.

B.2 Access Rights for Traffic

The second component of each envelop-spec entry in the filter envelop is the access rights field. It specifies what operations a delegate can perform on a traffic flow, once it has defined a filter for that flow. We distinguish between QoS-related operations and packet processing operations.

Once a filter has been installed, the delegate can assign this filter to a node in the resource tree. The data flow that corresponds to this filter will then receive a specific QoS, e.g., the bandwidth guarantees allocated to the node. To perform this operation, the delegate must have the (q) permission bit set, which represents the ability to provide Quality of Service, in the access rights field of the envelop-spec that covers the filter. Recall also that one of the access rights on a resource node is (u), which means that the delegate can use the node to serve traffic flows. This means that before a delegate can make a reservation, two conditions must hold: (1) the filter for the traffic must conform to an envelop-spec entry for which the delegate holds the (q) permission, and (2) the delegate has (u) permission for the resource tree node that represents the allocated bandwidth.

A delegate can also enable packet processing for a traffic flow by specifying that the flow should be processed by a specific pre-installed packet processing module (PPM), as described in Section II. This is done by instantiating a specific PPM instance and assigning a filter to that instance. By doing so, the flow corresponding to that filter will be processed according to the specification of the PPM instance. For example, a delegate can create a PPM instance that will tunnel packets to a specific IP address. This is done by creating an instance of the tunneling PPM and specifying the exit point of the tunnel as a parameter.

The range of packet processing operations that can be performed is very large. Given this diversity, it is at this point unclear what would be the best access rights abstraction to control the broad range of operations in a meaningful way. As a first cut we categorize the possible packet processing actions into the following groups:
1. Actions that have only local effects and do not change the content of the packets being forwarded, e.g., selective packet dropping, and DiffServ related modules. We use (l) to represent this type of processing.
2. Actions that alter the packets, but do not affect the normal forwarding of packets. This set of actions are often relatively heavy weight, e.g., encryption, compression, or special data transcoding. We use (a) to represent this type of processing.
3. Actions that have network-wide effect, in that they can change the normal packet forwarding. These actions include various redirecting methods, for instance, redirection can be done by specifying a different output interface, or by using IP-in-IP tunneling. We use (r) to represent this type of processing.

Clearly, for some of these actions, it may not be sufficient to limit only the action, but it may also be necessary to limit the scope of the action. For example, for the tunneling action, we must be able to restrict where packets are tunneled. This is done by including a parameter range in the ACL as an additional parameter, and by checking the parameters whenever a delegate creates a new packet processing instance. Clearly, the type of the parameters, and the precise nature of the checking will depend on the processing module. This solution is not completely general, but it is sufficient for the Darwin system, which supports only pre-installed, trusted data plane extensions.

TABLE II
Example Access Control List for traffic protection

delegate-id	traffic-spec-id	access rights	parameters
11	1	`l--`	
11	2	`-a-`	
11	3	`l-r`	if:fxp0
11	4	`--r`	dest:128.2.205.111

Table II shows some example access control lists used to control delegate 11. It can take "local" actions on flows conforming to the traffic-spec 1; it can take "alter" actions on flows conforming to the traffic-spec 2; it can take "local" and "route" actions on flows conforming to the traffic-spec 3; and the outgoing link must be interface fxp0; it can take "route" actions on flows conforming to the traffic-spec 4, and the destination must be 128.2.205.111.

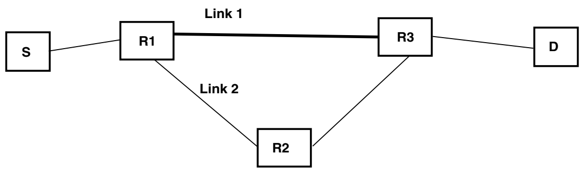

Fig. 5. Example application network topology.

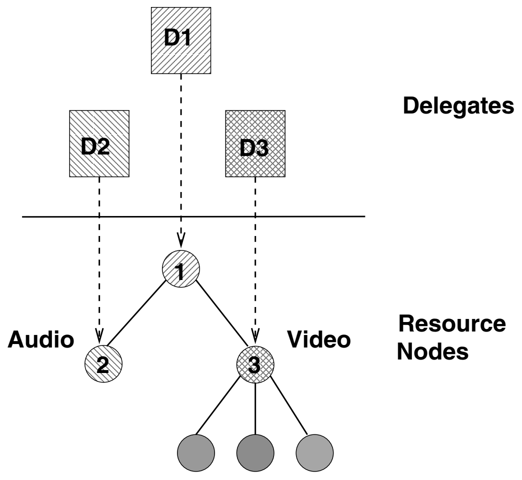

Fig. 6. Resource subtree on Link 1 and the association with delegates.

C. ACL Usage Example

We illustrate the usage of access control list with a simplified video-conferencing example(Figure 5). The sender

S of the application streams audio and video data through a simple network that consists of three routers, R1, R2 and R3, to the receiver D. The normal route from S to D goes through R1 and R3 only. Link 1 is the bottleneck link that often gets congested. A certain amount of bandwidth on Link 1 is allocated to this application represented by Node 1 in Figure 6. A delegate D1 is sent to router R1 to perform dynamic resource adaptation.

D1 has full control of the bandwidth allocated to this application, i.e., it has `cmdrnu` rights on Node 1. The filter envelop assigned to D1 limits the traffic that it can control. These traffic flows must have source address S, destination D and their port numbers must be one of the port numbers used by the audio or the video data. D1 can then subdivide the bandwidth by creating two nodes, Node 2 and Node 3, for the audio data and video data respectively. D1 allows delegate D2 to take charge of the audio traffic and delegate D3 to be responsible for the video traffic. D2 is assigned with `----nu` rights on Node 2 and `--r` right on the audio flow (defined by the audio ports). D3 is assigned with `cmdrnu` rights on Node 3 and `la-` rights on video flows (defined by the video ports).

When delegates and the access control lists are set up, the sender can start sending data. When congestion occurs on Link 1, D2 and D3 take proper measures to provide gracefully degraded quality. For example, D2 can reroute the audio traffic to Link 2, since it has "`r`" (route) right on the audio flow. As a result, the receiver can still receive sender's voice while the video quality may be bad. With "`c`" (create) right on Node 3, D3 can divide the bandwidth devoted to the video stream among the different types of video frames, e.g., B, P, I frames in MPEG. Since D3 has "`la`" (local, alter) rights on video traffic, it can selectively drop certain sub-flows or even perform other operations such as transcoding or compression to provide the best video quality possible. Other delegates in the system cannot use the bandwidth allocated to the video-conferencing application or manage its traffic since they do not have the necessary access permissions.

V. External Policy Checking

In the previous section we described an access control mechanism that can be used to protect link bandwidth and data traffic on programmable routers. Using a set of access control lists, the router can restrict the access of each delegate in an efficient way by checking the parameters of each RCI call. As described in Section III, for more complex operations, specifically the creation of new delegates or the transferring of access rights between delegates, we rely on an external policy manager. These operations are more complex because they require the creation of new access control lists, or changes to existing access control lists. In this section, we discuss how we can verify the correctness of these operations, focusing on delegate creation. Other operations such as the transfer of access rights can be handled similarly. We first look at the case where an external delegate initiator wants to create a delegate and we then look at the case of local delegate creation on the router.

A. Policy Manager

We assume there is one Policy Manager (PM) operated by a trusted authority within a domain. It has full control and knowledge of the routers and links of this domain. A network manager will typically be responsible for defining the policy for delegates in the network using a policy definition language. Users within the domain must register with the PM before trying to install delegates on the routers of this domain.

The primary task of the PM is policy enforcement, i.e., assigning access permissions to delegates based on the policy specified by the network manager. Besides acting as a policy server, the PM also serves as the authentication center and session key distribution center in the security protocol we present below. The protocol allows policies and delegate code segment to be transferred to routers securely by guaranteeing message integrity and confidentiality, and it also allows the PM and routers to authenticate users.

B. Access Policy Description

The PM decides on an access policy for a user based on the identity of the user and the operations the delegate would like to perform on the selected router. An access policy must specify the following information:
1. The amount of resources, e.g., bandwidth, that can be allocated to the delegate.
2. The filter envelop that defines the traffic that can be accessed. For example, the PM may only allow a user's delegate to install filters that have a source address field equal to the user's host address. This way, the PM limits the delegate to only access traffic that is initiated by the original user.
3. Access rights that regulate operations on bandwidth. For example, the PM may only allow a delegate to monitor the bandwidth usage of a resource node and disallow other operations on the node.
4. Control over traffic processing. The PM may specify what type of processing modules can be applied to certain flows. For the processing that has network-wide effect, the PM must specify some extra parameters to prevent network-wide security violations. For example, the PM specifies the acceptable encapsulation source and destination addresses for tunneling. The PM determines this information based on the concept of a "virtual mesh", which identifies all the network resources a user is allowed to use [5]. The idea is that users should only be allowed to redirect traffic within their virtual mesh, so that they cannot affect other parts of the network.

C. Access Policy Transfer

The communications between the users, PM and routers (Figure 3) must be secure to ensure the security of the system. We designed a secure communication protocol based on Kerberos [19], [18] that allows (1) users to authenticate with routers through the use of the Policy Manager; (2) secure transportation of delegate code and corresponding access policy information to routers.

C.1 Protocol Description

We use DI, delegate initiator, to represent the user who wants to install a delegate. Suppose there are n DIs in the domain, and we number them DI_1, ..., DI_n. We assume that DI_i shares a secret key with PM, K_{DI_i}. Suppose there are m routers in the domain and they are numbered R_1, ..., R_m. We assume R_j shares a secret key with PM, K_{R_j}. We examine the case of DI_i installing a delegate on R_j. The protocol contains the following message exchanges in order.

1. $DI \to PM$**:**
$DI_i, K_{DI_i}(R_j, N_{DI_i}, L)$
where: DI_i is the identifier of delegate DI_i; R_j is the identifier of target router R_j; N_{DI_i} is a random sequence number chosen by DI_i as a nounce to prevent replay; and L is a "proposal" that contains the list of resources that the delegate to be installed will access on router R_j.

This message serves three purposes. First, this is the way that DI_i authenticates itself to the PM. Upon receiving this message, PM uses the key corresponding to DI_i to decrypt the second half of the message. If the message is successfully decrypted, the PM is then sure about the authenticity of the message and the freshness can be verified with the nounce. Second, DI_i uses this message to request a session key to communicate with R_j; Third, DI_i requests an access policy, i.e., the set of access permissions for the delegate it is installing.

After receiving the above message, the PM executes the following steps: (1) the PM creates a globally unique id for the delegate to be installed DI_{ijk}, which means the kth delegate DI_i creates on R_j; (2) the PM generates a session key K_{ij} for communication between DI_i and R_j; (3) the PM produces an access policy, P, for this delegate based on the identify of DI_i and the proposal L. It then sends the following reply message.

2. $PM \to DI$:
$K_{DI_i}(K_{ij}, R_j, DI_{ijk}, P, N_{DI_i}+1)$,
$K_{R_j}(K_{ij}, DI_i, DI_{ijk}, P, N_{DI_i}+1)$
This reply message has two parts. Part 1 is encrypted with DI_i's secret key. DI_i decrypts it to retrieve the session key, the id for the delegate, the access policy and a nounce. Part 2 is encrypted with R_j's secret key; it contains the session key, delegate id and the policy. DI_{ijk} and P together are called a **credential**.

3. $DI \to R$:
$K_{ij}(del-code, DI_i, DI_{ijk}, P, N_{DI_i}+1)$,
$K_{R_j}(K_{ij}, DI_i, DI_{ijk}, P, N_{DI_i}+1)$
This message also has two parts. The first part is the DI's id, the delegate code, the credential and the nounce encrypted with the session key. For performance concern, the code part can be replaced with a message digest and the code itself can be sent in clear message if no secrecy of the code is required. An alternative to including the code with the request is to replace it with a reference to a secure code server. The second part is the same as the second part in the previous message.

When R_j receives the message, it decrypts the second part of the message to reveal the shared key K_{ij} and the credential for this delegate. At this stage, R_j believes that this part of the message is from PM because PM is the only other entity that knows K_{R_j}. R_j uses key K_{ij} to decrypt the first part of the message. If successful, R_j now knows that this message is from DI_i, since it is the only party, other than the PM, that knows K_{ij}.

The DI identifiers and the sequence numbers in these two message parts must match to prevent tampering and replay attack.

4. $R \to DI$:
$K_{ij}(confirm-msg, DI_i, DI_{ijk}, N_{DI_i}+2)$
R_j sends this message to DI_i to report the outcome of the delegate installation: success or failure.

C.2 Router Security Manager

The Router Security Manager (RSM) is the entity on the router that is the endpoint for the above protocol. Once the RSM has verified that the delegate creation request is valid, it will ask the delegate runtime environment to create the delegate, it will set up the ACLs as specified in the request, and it will pass the necessary handles (which can identify flows and resource nodes) to the delegate so it can perform its tasks. Any RCI call made by the delegate can then be checked by the ACM based on the ACLs.

C.3 Protocol Discussion

From authentication and session key establishment point of view, the above protocol is essentially the same as Kerberos V4: DI is equivalent to the client in Kerberos, Router corresponds to the server, and the PM acts as the Ticket-Granting Server(TGS). We do not have a separate Authentication Server since we assume the shared key between every DI and PM, and the key between every router and the PM are established using an external mechanism. Analysis of Kerberos's correctness and effectiveness properties can be directly applied to our protocol. This protocol can be extended to support delegate installation in multiple domains.

There are two ways to transport the access policies assigned to delegates to a router. Typically, users retrieve policies from the PM, bundle them together with the delegate code segment or a reference to a secure code server, and transfer the whole package to the router using the above protocol. The advantage of this approach is that it may be possible to optimize the policy checking on the PM by combining the requests for multiple routers. Alternatively, the router and the PM may interact directly. This will for example be necessary to check the correctness of a complex delegate action on the router, or if the PM wants to revoke the access rights of a specific user. For interactions between routers and the PM, a protocol such as COPS [3] could be used.

D. Local Delegate Creation

In the first part of this section we discussed how an external delegate initiator can create a delegate. We now briefly look at the case where a delegate on the router wants to spawn a "child" delegate. There are two cases to consider: (1) the child delegate will only access the resources for which the parent delegate has permissions, for example, a child delegate is created to monitor sub-flows; and (2) the child delegate may need access to resources that the parent delegate does not have access permissions for, for example a signaling delegate creates a new delegate for a new user.

For case (1), the parent delegate can simply submit the child delegate code, the description of the desired resources to be accessed by the child delegate, along with its delegate ID to the RSM. The RSM can make a local decision to install the child delegate if and only if the child delegate's access profile is within the parent's profile. The RSM then generates a new unique delegate ID and updates the policy database by adding appropriate access control entries for the new delegate.

For case (2), since the child delegate needs to access resources that the parent delegate does not currently have permissions for, the RSM cannot make a local decision of whether or not to accept the child delegate. Instead, the RSM forwards the request to the remote PM. The RSM will install the delegate only when it receives the PM's approval. An alternative approach for this kind of delegate creation is to let the parent delegate contacts the PM itself for clearance of the request. In the takes, the role of the parent delegate is similar to that of an external delegate initiator.

VI. System Implementation

We implemented the described security mechanism in the Darwin system [13]. The Darwin kernel implementation is based on FreeBSD 3.4-R patched with the ALTQ framework release 2.1 [8] for the output port packet classification and scheduling. Darwin also supports several packet processing modules, including tunneling, rerouting, VPN encapsulation and encryption. DiffServ related modules are under development in the research group.

The RSM is implemented in Java, and it currently supports the creation of delegates written in Java. The RSM handles all external requests for bandwidth reservations and delegate creation. If an incoming request contains a request for bandwidth, RSM calls the local resource manager (LRM) to reserve the necessary bandwidth for the delegate (i.e., adding a node to the resource tree). If request includes a request for the creation of a delegate, the RSM forwards the request to the delegate runtime environment. It then writes the access rights specified by the policy manager into a policy file. The RSM keeps one file for each interface that the delegate operates on. The file name contains both the delegate's ID and the interface name, e.g., `del-policy.11021.fxp0`, where `11021` is the delegate ID and `fxp0` is the interface that this delegate operates on. This scheme allows the ACM to locate the access control list from the right file quickly.

The policy description is in ASCII text format and contains the following sections: filter envelop, which defines the possible flows can be accessed and the permissions; and the resource ACL, which lists the access permissions the delegate has on resource nodes.

The RCI is implemented as a Java library that delegates can call. This library simply forwards the RCI call to the LRM (Figure 2), which is implemented as a user-level process, called the admission control daemon (ACD). The RCI library includes the delegate identifier in the request that it forwards to the ACD. The Access Control Manager (ACM) is implemented as part of the ACD, and it intercepts each RCI call made by delegates. The ACM uses the delegate identifier and the interface the delegate operating on to find the proper policy file for access checking.

The implementation of the security protocol between users, policy manager and routers is still work in progress. It uses Java 2's security packages to do key generation, exchange, and message encryption and decryption.

VII. Related work

Several research groups have studied the security issues in active and programmable networks. The DARPA-sponsored Active Net Node OS working group proposed a secure architecture [12] for active nets. Nodes in active nets are active (AN) and support an execution environment (EE) that can execute active applications (AA), which can be active packets or active extensions. The secure architecture document highlights the components required in such a system and focuses more on traditional notion of security such as authentication, authorization, and cryptography. The Seraphim project [4] implements a flexible security architecture for active nets that can accommodate a wide range of security policies on an active node. The SANE project [2] addresses the active node security problem starting from a machine level, in which it guarantees a node starts operation in a trusted state and guarantees the system in a trusted state by applying dynamic integrity checks at the system runtime. SQoSH [1] is an architecture that incorporates SANE into the Piglet operating system to provide controlled access to allocations of system resources in an active network element.

Compared with the above systems, which generally present fairly broad security frameworks, the work presented in this paper is more focused. We specifically looked at the question of how to use access control lists to efficiently protect two types of resources (bandwidth and data traffic) that play a central role in routers, but that are typically not managed explicitly by traditional (endpoint) operating systems. We also focus on networks that use active extensions, while most of the above projects consider both extensions and capsules.

A framework for policy-based admission control [23] in IP networks that support Integrated Services or Differentiated Services is proposed in the IETF. The basic question it addresses is when admitting a request for resource from a signaling protocol, the decision should be made based not only on physical capacity but also on a policy. The framework relies on a remote authority, Policy Decision Point (PDP) to make policy decisions and a local entity, Policy Enforcement Point (PEP) to enforce the PDP's decision: either reject or accept the request. When compared with our programmable network system, in this framework, the role of PDP is similar to PM, and the role of PEP is similar to RSM. However both the PM and RSM are more powerful since we are dealing with running active extensions rather than a simple resource reservation request. PEP and PDP communicates directly via a protocol called COPS [9], [14]. In our scheme, RSM has the ability to communicate with PM directly for policies, but in most cases, users who send delegates retrieve policies from PM and send them to RSM. This results in a more efficient system in that it reduces the labor of the RSM.

Java 2 [16] uses a scheme based on a Public Key Infrastructure (PKI) to allow a machine to authenticate foreign code. While this design suits the web environment where Java is targeting, we choose a shared private key scheme for the authentication between users and routers. Shared key system is in general more efficient than PKI especially within a trusted domain, where each user is under the management of the system administrator. The choice of shared key authentication also eliminates the non-trivial problems related to public key distribution. In our system, we combine the functionalities of authentication, key generation with policy generation into one entity PM.

VIII. Conclusions

One of the fundamental concerns in programmable networks is that its programmability makes it subject to a wide range of security attacks. Active extensions, or delegates, can be dynamically injected into routers for execution. While these extensions can extend the router's functionality on the fly, e.g., to deploy new network protocols or services, malicious or faulty extensions can pose serious security threats to the routers. In this paper, we presented a set of security mechanisms that limits the access of extensions to the router's data plane resources.

The design is based on the observation that, unlike traditional operating systems, the unique resource access concerns in a programmable router center around access to link bandwidth and data traffic. The access policy that defines the set of access permissions to router resources, is assigned by a trusted policy manager (PM). The policy is then transferred and stored on the router, on which the delegate will be running. Whenever a delegate accesses a resource, the router checks the set of access permissions, and thus prevents any possible security violation. Our security protocol between users, routers, and the policy manager provides a secure channel for these entities to authenticate and communicate with each other. We implemented the security mechanisms in the Darwin system.

References

[1] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, Steve Muir, and Jonathan M. Smith. Secure Quality of Service Handling: SQoSH. *IEEE Communications Magazine*, 38(4):106–112, April 2000.

[2] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, and Jonathan M. Smith. A Secure Active Network Architecture: Realization in SwitchWare. *IEEE Network Special Issue on Active and Controllable Networks*, 12(3):37–45, May/June 1998.

[3] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) Protocol. IETF Request for Comments 2748, January 2000.

[4] R. H. Campbell, Z. Liu, M. D. Mickunas, P. Naldurg, and S. Yi. Seraphim: Dynamic Interoperable Security Architecture for Active Networks. In *Proceedings of OPENARCH 2000*, Tel Aviv, Israel, March 2000.

[5] Prashant Chandra, Allan Fisher, Corey Kosak, T. S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. In *Sixth International Conference on Network Protocols*, pages 177–188, Austin, October 1998. IEEE Computer Society.

[6] Prashant Chandra, Allan Fisher, and Peter Steenkiste. Beagle: A resource allocation protocol for an application-aware internet. Technical Report CMU-CS-98-150, Carnegie Mellon University, August 1998.

[7] Prashant Chandra, Allan Fisher, and Peter Steenkiste. A Signaling Protocol for Structured Resource Allocation. In *IEEE INFOCOM'99*, pages 522–533, New York, March 1999. IEEE.

[8] Kenjiro Cho. AltQ Framework. http://www.csl.sony.co.jp/person/kjc/software.html.

[9] D. Durham and et. al. The cops protocol, January 2000. Request for Comments 2748.

[10] Jun Gao, Peter Steenkiste, Eduardo Takahashi, and Allan Fisher. A Programmable Router Architecture Supporting Control Plane Extensibility. *IEEE Communications Magazine*, 38(3):152–159, March 2000.

[11] Jun Gao, Peter Steenkiste, Eduardo Takahashi, and Allan Fisher. A Programmable Router Architecture Supporting Control Plane Extensibility. CMU technical report, CMU-CS-00-109, March 2000.

[12] DARPA AN Security Working Group. Security Architecture for Active Nets. ftp://ftp.tislabs.com/pub/activenets/secarch2.ps, July 1998.

[13] The Darwin group. Darwin code release. http://www.cs.cmu.edu/∼darwin.

[14] S. Herzog and et. al. Cops usage for RSVP, January 2000. Request for Comments 2749.

[15] Keng Lim, Jun Gao, Eugene Ng, Prashant Chandra, Peter Steenkiste, and Hui Zhang. Customizable Virtual Private Network Service with QoS. Carnegie Mellon University, To appear in Computer Networks.

[16] Sun Microsystems. The Source for Java(TM) Techonology. http://www.javasoft.com/.

[17] George Necula and Peter Lee. Safe Kernel Extensions Without Run-Time Checking. In *Proceedings 2nd Symposium on Operating Systems Design and Implementation (OSDI'96)*, pages 229–243. Usenix, October 1996.

[18] B. C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32:33–38, Sepetember 1994.

[19] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of Winter USENIX Conference*, pages 191–201, 1988.

[20] Ion Stoica, Hui Zhang, and T. S. Eugene Ng. A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. In *Proceedings of the SIGCOMM '97 Symposium on Communications Architectures and Protocols*, pages 249–262, Cannes, September 1997. ACM.

[21] Eduardo Takahashi, Peter Steenkiste, Jun Gao, and Allan Fisher. A Programming Interface for Network Resource Management. In *1999 IEEE Open Architectures and Network Programming (OPENARCH'99)*, pages 34–44, New York, March 1999. IEEE.

[22] David Tennenhouse and David Wetherall. Towards and active network architecture. *Computer Communication Review*, 26(2):5–18, April 1996.

[23] R. Yavatkar, D. Pendarakis, and R. Guerin. A framework for policy-based admission control, January 2000. Request for Comments 2753.

[24] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource Reservation Protocol. *IEEE Communications Magazine*, 31(9):8–18, September 1993.