Secure Service Signaling and fast Authorization in Programmable Networks

Michael Conrad, Thomas Fuhrmann, Marcus Schöller,
and Martina Zitterbart

Institut für Telematik
Universität Karlsruhe, Germany

Keywords: Programmable Networks, Flexible Service Platforms,
Secure Signaling

Abstract. Programmable networks aim at the fast and flexible creation of services within a network. Often cited examples are audio and video transcoding, application layer multicast, or mobility and resilience support. In order to become commercially viable, programmable networks must provide authentication, authorization and accounting functionality. The mechanisms used to achieve these functionalities must be secure, reliable, and scalable, to be used in production scale programmable networks. Additionally programmable nodes must resist various kinds of attacks, such as denial of service or replay attacks. Fraudulent use by individual users must also be prohibited.
This paper describes the design and implementation of a *secure, reliable, and scalable* signaling mechanism clients can use to initiate service startup and to manage services running on the nodes of a programmable network. This mechanism is designed for production scale networks with AAA-functionality.

1 Introduction

Programmable and active networks extend programmability of network components from the network's edge into the network itself. (See [2,3] for an overview of the basic concepts). Among its key motivations is the idea to quickly and flexibly create new services within a network. This could overcome the long deployment-cycles usually experienced in this area. In such programmable networks, services are created by so-called *service modules*. These are executed in an *execution environment* on the *programmable nodes* of the system (e.g., [8]). The user can start such service on demand to support its already running application.

Varying from approach to approach, programmable nodes are expected to be deployed densely or sparsely. The FlexiNet project (www.flexinet.de), in the context of which this work has been performed, assumes that the programmable nodes are placed near the network edge, e.g. as gateway of a (small) sub-network or as additional programmable nodes within such a sub-network. Typical locations might thus be the access routers of wireless networks, small offices, home offices, or off-path programmable nodes at the Internet service provider or customer premises. These off-path nodes provide supplementary resources for services the on-path routers provide. The service startup process in general is as follows: first access rights of the client to start the service

must be checked and then an evaluation process [8] selects the node where the service gets executed. The access rights of a client are checked by an authorization server which can be found dynamically by the client through a indirect signalling scheme using any one programmable node on path between client and content server (see section 2). After successful authorization the evaluation process determine the programmable node on which the service gets executed – depending on the service this can be the on path node or any off path node. For this paper we will assume that the service gets executed at the programmable node which is involved in the authentication process.

In todays networks it must be assumed that the traffic between the clients, the programmable nodes, and the service providers could be intercepted and spoofed by malicious devices at will. Since programmable network nodes are enhanced routers, their availability is critical for the overall connectivity of clients in the sub-network behind such a node. A programmable node usually handles data streams for multiple receivers. This makes a programmable network node an attractive target for any kind of attack. Hence, special care has to be taken to assure robustness and stability.

The second goal to be achieved is user authentication, authorization, and accounting (AAA). A service provider might want to offer different kinds of services to different user groups. Some of the services should be accessible only by local users, other services may be used by roaming users, too. Services might require a fee, and therefore accounting information must reliably be collected by the provider. This means that an attacker must not be able to forge its identity in order to charge the costs to another user or to start services he does not have access permissions to.

Scalability of the signaling scheme is the third goal of the design. The mechanisms used must cope with multiple signaling messages from many users in parallel. The scalability of the system depends on the resources that handle requests and how these can be duplicated and distributed over the network. The presented design will show on the one hand that only little resources will be needed at the programmable node's side to handle requests and on the other hand that we are willing to overwhelm the authentication and authorization server to preserve the robustness and stability of the programmable node.

This paper presents a secure service signaling mechanism that allows the reliable operation of a programmable network under regular conditions and attacks. Section 2 presents the design of our approach and a threat analysis, followed by implementation details of the proposed mechanisms in section 3, and section 5 finally concludes with the summary.

2 A flexible signaling concept

Generally, there are four types of entities in the programmable network scenario we examine. Without loss of generality we assume that there is a client, a server, a programmable node, and a service module repository (see fig. 1). The client receives a data stream from the server and wants to use a special service provided by the programmable network. The service modules that provide this service are stored in the service module repository. From there they are loaded onto the programmable node. To start the service

the user sends a so-called *service start request* to the programmable network. This can be done either directly or indirectly.

Direct signaling can only be used if the client knows one or more programmable nodes, e.g. by using a dynamic configuration protocol like DHCP. Static configuration of clients might also be a solution in some scenarios. With direct signaling the service start request can then be sent directly to one of the programmable nodes.

Indirect signaling applies when the client has no knowledge about any programmable node. In this case, the client simply sends its service start request towards the server's address. Any programmable node that supports indirect signaling must filter transiting packets to discover service start requests. In this way a indirect signalling packet is filtered by the first programmable node on the path between client and server. Such a filter is easily implemented as a programmable network service as shown in section 3.

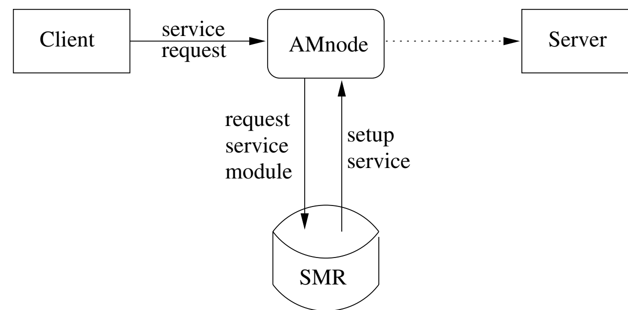

Fig. 1. Service deployment

The minimum of information that must be contained in a service start request is a service identifier. How these identifiers are assigned to the services, is outside the scope of this paper, but we assume that the client knows the identifier, which is associated with the service it is about to start. Furthermore, we assume that the client's address is included in the service start request, too. It is used to notify the client of the success or failure of its request.

This simple and flexible approach suffices for a client to set up a service. Beyond the basic signaling exist several issues of the network provider concern e.g. to allow accounting: *Who is requesting a service in the network? Is that person authorized to do so?* And to secure its network infrastructure: *How can the programmable nodes be protected from spoofed service start requests? How can replay attacks be prevented?* These questions will be addressed in the following sections:

2.1 A secure approach

To design a secure system, some assumptions about trust relationships of the involved parties must be clarified. The service provider runs the programmable network nodes,

the service module repositories, an authentication server, and an authorization server. Both servers are fully trusted by the programmable nodes and the repositories.

The trust relationships of the client are more complicated, especially if clients can roam between multiple domains. To trust a repository, the client must be able to authenticate the repository. After a successful authentication, the client will bind a session key and other temporal data to the proven identity of the repository, and only messages authenticated with that session key are accepted. Since repository and programmable node implement a full trust relationship, the client extends its trust of the repository to all programmable nodes in the domain of this repository. A programmable node proves its domain membership by authenticating its messages to the client through the knowledge of the session key. After terminating the session, all temporarily data gets deleted, and no further messages authenticated with that key are accepted any more.

On the other hand, the repository needs to authenticate the client to grant or deny access to its services. Since clients may roam to the domain of the repository, a long term secret between these two entities can not be assumed. A PKI-based authentication scheme provides secure authentication with good scalability. For free services, such a scheme is sufficient but not for services that require a fee. Depending on the economic relationship between the service provider and the client, accounting information must be available to the repository. This might imply secure communication between the repository and an accounting server in the home network of the client. The client must provide information about its accounting server to the repository during authentication to allow online checking of e.g. available credit.

The protection of the programmable nodes from attacks is the top priority. To this end, we propose the paradigm that a programmable node only handles authenticated requests and does *not establish any state for unauthenticated requests* on the node. On the other hand, for administrative reasons, user authentication and authorization data should not have to be distributed to every programmable node. We accomplish these two fundamental requirements by combining both the authentication scheme (used for initial client contacts) with a special request redirection mechanism. This mechanism scales very well with the number of clients since a programmable node can redirect messages up to 100 MBit/s in real-time.

Basic constraints As described above, we do not assume any pre-shared secret between the client and the programmable network provider. However, since a pure certificate-based approach suffers from the much greater computational overhead of public key cryptography, as compared to a hash function or secret key cryptography, the use of asymmetric cryptographic algorithms should be limited as much as possible. Experiments on hand-held devices like a Palm (20 MHz - Dragonball) were presented in [5]. They show that only 0.14 RSA sign operations can be performed per second, rendering a application based on such operations unusable.

2.2 Authentication scheme

Our proposed authentication scheme uses public key algorithms to prove the client's identity to the network provider, to prove the identity of the service provider to the

client, and – at the same time – to distribute the keys for the HMAC algorithm that is later used for the authorization scheme and client to service communication. The process is as follows:

The client generates an authentication request containing its identity together with its certificate and the socket (IP-address and UDP port number), at which he will listen for the response. Before sending this request to the programmable node the client creates a MAC (message authentication code) for the message. This is a digitally signed hash value of the message, and is appended to it. Thereafter, the request is sent to a programmable node by direct or indirect signaling.

The programmable node forwards the request to the authentication server using the request redirection mechanism, where the identity of the client is checked. If the authentication fails, no error message is sent to the client, to prevent the system from responding to flooding attacks. Otherwise the authentication server adds the session key, a key lifetime, two sequence numbers, its certificate, and the IP-address of the authorization server to the client message to create the response. The session key is encrypted with the client's public key and the complete authentication reply is digitally signed by the authentication server. The response is sent to the IP address and the port number of the client contained in the request.

If the authorization server is not collocated with the authentication server the session dependent data must be transferred to the authorization server, too. Therefore, the authentication server sends the session key and the two sequence numbers to the authorization server in an additional message. This message must be integrity protected and confidentially transferred to the authorization server.

The client validates the server's certificate first, either on its own or by an online protocols like OCSP [11]. Then it checks the signature and finally decrypts the session key and stores the sequence numbers.

Message (1), (2) and (3) in fig. 2 are the message of the authentication scheme.

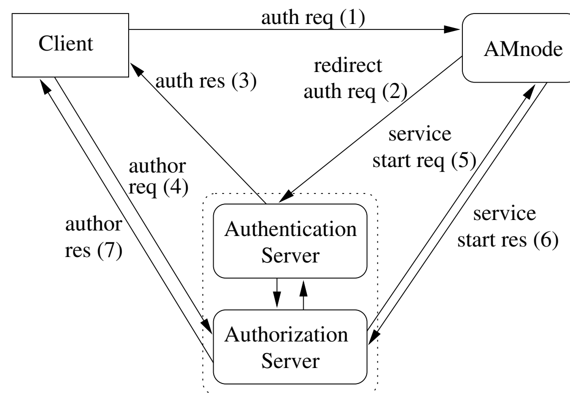

Fig. 2. The secure signaling scheme

Threat analysis If an implementor uses state-of-the-art encryption (e.g. RSA) and hash function (e.g. SHA1) it is reasonable to assume that attacks on these will not be successful. Other kinds of attacks must be analyzed in more detail: DoS attacks and replay attacks.

Since the programmable node only forwards authentication requests to the authentication server without establishing any local state, the resistance against DoS attacks solely depends on the address rewriting functionality. As shown in [4] our system can handle simple UDP header manipulations in real time up to 100 Mbit/s. Therefore, the authentication server might become a bottleneck if DoS attacks are launched at the system. Since the authentication server checks the authenticity of any request, a flooding attack might lead to CPU exhaustion of the server. No further real requests can be handled at the authentication server, thus preventing the authentication of new users. It is the intention of the design to sacrifice the authentication server in favor of the programmable nodes. To limit DoS attacks during authentication, an additional cookie exchange or installment of filters at the programmable node might further reduce the impact of DoS attacks. Such filter installment and configuration is still a subject of our research and is not further covered here.

The second type of attack is the replay attack. The attacker monitors the messages exchanged between client and programmable node, and replays these messages sometime in the future. This attack can be prevented if the authentication server creates different session keys for every request. An attacker gets a positive authentication response from the authentication server but can not replay any further sniffed messages, because these messages can not be authenticated. To prevent the attacker from replaying the authentication request infinitely until the authentication server picks the same session key by chance, a monitoring facility at the authentication server should log all authentications, and might disable the authentication of users in case of such an attack for a limited time.

Authentication result All further communication between the client and the programmable network can be protected with the now established shared secret. During further communication, the sequence numbers are incremented by the sender and checked by the respective recipient to be in increasing order. Any message with a smaller sequence number is silently discarded. Afterwards the recipient checks the authenticity of the message. If this check succeeds, the receiver stores the sequence number of this message as the new lower boundary; otherwise the message is silently discarded. This mechanism prevents replay attacks of sniffed messages. (We accept that messages are discarded and have to be retransmitted with a new sequence number if messages sent with correct sequence numbers got reordered during transmission.)

2.3 Authorization scheme

After a successful authentication the client can request one or more services to be started on the programmable net. The authorization server checks the access rights of the user with respect to the requested service. As with the authentication server, the authorization server can be a stand-alone server or be collocated with the service module repository.

The client creates an authorization request, uses the session key to generate a MAC and appends the MAC to the message. Besides the user ID and the sequence number, this request contains the service ID and the service parameters (if necessary). The message is then sent to the authorization server indicated in the authentication response (message (3) in fig. 2). This server first checks the sequence number, then the MAC, and finally processes the request, if all checks have succeeded.

If the user is not allowed to start this service, an error notification is sent to the client, indicating why the access was denied. Otherwise the authorization server informs the programmable node that the service can be started (message (5) and (6) in fig. 2). After a successful service startup on the programmable node, an authorization response is sent to the client (message (7) in fig. 2). This response carries any necessary information to allow client to service communication: IP address of programmable node, id of the service on that node, and two sequence numbers.

The client derives a new service session key for this service by using a hash function with the authorization session key and the two new sequence numbers from the server as arguments. To allow secure client to service communication, the corresponding session state containing this session key, and the sequence numbers have to be transfered to the programmable node, too. The server derives the service session key in the same way and sends the key to the programmable node in an extra message, which can be piggybacked to message (5). The client to service communication is protected using the service session key and the sequence numbers. Section 2.4 shows how the keep-alive messages are protected by this key.

Rekeying The introduction of sequence numbers, to protect the communication between client and programmable node against replay attacks, makes a mechanism to handle the wrapping of these sequence numbers necessary. As soon as a sequence number cannot be increased, new session keys must be requested from the authorization server. Either the client sends an authorization request to the authorization server, as described above, containing additionally the service dependent data to inform the server for which service new session keys are requested or the programmable node sends a rekey request containing the same data. The authorization server generates a new session key and transfers this key plus the two new sequence numbers for this security association to the client and the programmable node.

Threat analysis The authorization scheme is resistant against replay attacks through the usage of sequence numbers. During authentication two sequence numbers have been transferred to the client. The client uses the first one to authenticate messages sent to the server, the other one to receive message from the server. As long as the session key is known only to client and server, it is reasonable to assume that no attacker can create a valid MAC for a message with a valid sequence number.

Including the IP address of the authorization in the authentication response informs the client reliably to which authorization server the security association has been established. This information assures the address of the authorization server to the client, even if an attacker manipulated the unprotected IP header of the authentication response.

Third, the scheme has a good resistance against DoS attacks. Using the first message of the authorization scheme, the client must prove that it has access to the session key. The authorization server first checks the authorization request, and only in case of a valid request is the programmable node involved in the process. Again a DoS attack on the system might bring down the server, but the programmable node is not disrupted.

The creation of session state on the programmable nodes is delayed, until the client has proven access to the session key. This implies that session state on the programmable node is only created for actual running services on that node. As soon as the service is stopped, the node can purge any service dependent data.

Authorization result The requested service has been set up on the programmable node, and the user is informed that he can use the service. If client to service communication is supported by this service, all address information of the service is transferred to the client. Depending on the service, reconfiguration requests and keep-alive messages can be sent directly from client to the service instance.

2.4 Service lifetime

To explicitly support roaming users, we propose a soft state approach for services on programmable nodes. Since client to node communication can break down suddenly, the client cannot always send a service stop request to the node. Service execution and accounting of service usage must be stopped by other means. A service execution based on a soft state approach enables the desired behavior, but requires the client to refresh the state periodically. If no refreshment message reaches the node for a configured amount of time, the service is stopped automatically.

The message to refresh the state of the service must be protected against manipulation and replay attacks, too. The client uses the session key and sequence numbers to generate this request and sends the request directly to the programmable node the service is executed at. The mechanisms to check the message authenticity are performed, as during the authorization.

3 Implementation

Within the FlexiNet project we have implemented an exemplary client signaling GUI, the active node, and the service module repository. The authentication server and the authorization server are both collocated with the service module repository. Every programmable node creates a TLS tunnel [6] to its configured service module repositories during system startup. In contrast to the ordinary use of TLS, we demand mutual authentication of the communication peers. A programmable node uses the TLS tunnel to securely download service modules from the service module repository. Additionally, we will use this tunnel to exchange service start request and response messages between programmable nodes and the authorization server.

For user and machine identification we are using X.509 certificates, which carry RSA keys for the signature generation and validation. As cryptographic hash functions, HMAC/SHA1 are used in our extended signaling scheme.

This section details implementation issues of the message redirection mechanism, which is followed by an example of our message format.

3.1 Redirection mechanism

To filter signaling messages at a programmable node, a special service module – called signaling filter module – must be installed. It is started during system startup and runs in a special environment – the so called framework execution environment. This environment, and thereby the signaling filter service, cannot be stopped by normal means. The service is active as long as the programmable node is working. Furthermore, the signaling filter service is the only service allowed to forward messages to the programmable node's framework. The main task of the framework is node management, which includes service setup and termination. The framework instantiates a new execution environment every time a new service gets started on the programmable node, and the required service modules are loaded into this environment. If the required module is not stored in the local module cache, the framework has to download that module from the service module repository, via the established TLS tunnel.

The signaling filter service installs three network hooks during startup, in order to filter UDP ports 5000, 5001, and 5002. While the hook on port 5000 is accepting all bypassing traffic for any destination, the two other ports only accept messages directly addressed to the node. In our implementation the client uses port 5000 to indirectly send the authentication request to the programmable network. This allows for clients to discover a programmable node without knowledge about the network topology. The client sends its signaling request towards the server, and if a programmable node is located on the path between client and server, it will filter this signaling message. Additionally, port 5001 is used by clients to directly signal the authentication request. All other messages, like keep-alive requests, have to be sent to port 5002.

The messages, which a programmable node filters on port 5000 or 5001, will be redirected, without any processing, to a connected service module repository. To achieve this, the signaling filter service just has to replace the destination address in the UDP packet and to recalculate the UDP checksum, if used. Thereby, no state has to be established, and the node is protected against flooding attacks.

Any message filtered on port 5002 must be forwarded from the signaling service to the local framework of the programmable node. Here the message gets re-instantiated, and its signature is verified. Only messages authenticated with the session key are accepted at port 5002. All other messages get immediately discarded.

3.2 Signaling message format

In an abstract signaling class, three types of messages – request, response for synchronous communication, and trap for asynchronous communication – and the basic attributes of these signaling messages are defined. The basic attributes are : command, group, msg-id, user-id, client and node sequence number. Every implementation of a message must be derived from this abstract class and can add message dependent attributes. Therefore, the implementation must assign a name and a type to the attribute, and provide methods to get and set the attribute values.

Authentication request Fig. 3 shows a serialized authentication request, which always consists of two parts: a message part (line 02-07) and a signature part (line 08-10).

```
01 <flexinet version="2.6"> <signaling>
02     <request client-seq="0" command="authenticate" group="access"
03              msg-id="42" node-seq="0" user-id="client@flexinet.de">
04       <scalar name="ip-protocol" type="string"><string>udp</string></scalar>
05       <scalar name="udp-port" type="int"><int>5000</int></scalar>
06       <scalar name="ip-address" type="string"><string>192.168.0.1</string></scalar>
07     </request>
08     <signature algorithm="SHA1withRSA" msg-id="42" user="client@flexinet.de">
09       U1Qe1z/9drf75zFA7JH18AWalz/VTzaFmsFJX6g1WQYAEAWPtoXTM1d...
10     </signature>
11   </signaling> </flexinet>
```

Fig. 3. Authentication request message

The command `authenticate` within the group `Access` denotes that a client wants to authenticate itself to the system. The ID of the client is stated in attribute user-id. The two sequence numbers are not used during authentication and are set to zero. The id attribute is used to bind a signature to a request. This binding is necessary if multiple request, response, and trap messages are sent within a single signaling message. Besides these basic attributes, an authentication request contains attributes like the client's IP address, port number, and the type of the transport protocol.

The signature part states the algorithm used to generate the signature, which is bound to the message part with the same id. The signature is computed using the key of the denoted user. In the example above, `SHA1withRSA` is used as the signature algorithm. Since the object representation is unsuitable for signature generation, we have chosen the canonical XML serialization as input to the signature algorithm.

```
01 <flexinet version="2.6"> <signaling>
02     <request client-seq="3749" command="start" group="service"
03              id="46" node-seq="63953" user="client@flexinet.de">
04       <scalar name="ip-protocol" type="string"><string>udp</string></scalar>
05       <scalar name="udp-port" type="int"><int>5000</int></scalar>
06       <scalar name="ip-address" type="string"><string>192.168.0.1</string></scalar>
07       <scalar name="service-id" type="int"><int>23</int></scalar>
08       <scalar name="private-service-option" type="int"><int>3</int></scalar>
09     </request>
10     <signature algorithm="HMACSHA1" id="46" user="client@flexinet.de">
11       cvXzFkm6m2uc2NypaQ8Tbsai5RE=
12     </signature>
13   </signaling> </flexinet>
```

Fig. 4. Authorization request message

Authorization request To start a service, the client sends an authorization request (see fig. 4 to the authorization server. The message part denotes the command `start` within

the `service` group (line 02). Sequence numbers and message id are set according to the current client state (line 02-03). In addition to these basic attributes, the client specifies the desired service using the `service-id` attribute (line 07) and service parameters using the `private-service-option` attribute (line 08). To receive an authorization response the attributes `ip-address`, `ip-protocol`, and `udp-port` (line 04-06) are set as in the authentication request.

The signature is generated using the `HMAC-SHA1` algorithm and the shared secret. Again the id binds this signature to the corresponding message part.

4 Related work

The related work can be divided into three categories: signaling protocols, general authentication and authorization protocols, and authentication and authorization in programmable networking.

GIMPS [12] is a draft of a general signaling protocol. The authentication schemes incorporated assume that the client knows his communication peer, which is not always true for programmable networks.

In the area of general authentication and authorization protocols, many solutions for different scenarios have been proposed. EAP [7] is an extensible authentication protocol used for network access control. The protocol was designed to authenticate a dial-in user to the network access server. A prerequisite of the protocol is that the user authenticates himself towards the next hop, since EAP is a layer 2 protocol. This behavior is not applicable to programmable networking, since the programmable node might be multiple hops away from the user.

Kerberos [10] provides user authentication based on symmetric key algorithms. A user authenticates himself via user name and password, and uses tickets to authenticate himself towards the resources of the network. The drawback of Kerberos is that it cannot support roaming users moving into a Kerberos domain. In our opinion, support of wireless clients is essential for programmable networking.

Key exchange protocols always implement an authentication scheme, and additionally solve the key distribution problem. TLS [6] is a TCP-based security protocol including a key exchange. The modifications to the protocol, which would be necessary to support indirect signaling, are non-trivial. Nonetheless, some basic mechanisms of TLS, like server based key generation, are reused in our design. IKE [9] is another key exchange protocol. It is based on UDP and fits many needs outlined, but the key generation mechanism can only use the Diffie-Hellmann algorithm. Furthermore, the complexity of the protocol is a known drawback of IKEv1, but might improve with IKEv2, whose standardization should be completed in the near future.

Within the active and programmable networking area, only little research on secure signaling has been introduced. Like in SARA [1] an analysis of security aspects and possible solutions are provided, but some forms of attacks have been neglected, like DoS attacks on the active router. To verify a request in SARA, the active router must first download the module from the code server and then verify the authenticity of the request. An attacker can easily mount an DoS attack on the active router by requesting different modules each time.

5 Conclusion and Future Work

We have presented a secure and scalable signaling scheme for user authentication, service startup, and service management. The authentication scheme can be used for direct and indirect signaling in the case, that the nearest programmable node is not known to the client. The usage of asymmetric cryptographic algorithms is thereby limited to the authentication process and all further messages are protected by cryptographic hash functions. The presented signaling scheme resists active attackers and stems the threats of denial of service attacks.

A secure evaluation scheme and a secure service relocation scheme will be available shortly and presented in the near future. These schemes will be build from the same building blocks as the presented authentication and authorization scheme, keeping the protocol complexity modest, and thereby lighten the analysis of the protocol.

References

1. M Bagnulo, B. Alarcos, M. Calderón, and M. Sedano. ROSA: Realistic Open Security Architecture for active networks. In *Fourth Annual International Working Conference on Active Networks (IWAN)*, 2002.
2. Kenneth L. Calvert, Samrat Bhattacharjee, Ellen Zegura, and James Sterbenz. Directions in active networks. *IEEE Communications Magazine*, 36(10):72–78, October 1998.
3. Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, and Daniel Villela. A survey of programmable networks. *ACM SIGCOMM Computer Communication Review*, 29(2), April 1999.
4. M. Conrad, M. Schöller, T. Fuhrmann, G. Bocksch, and M. Zitterbart. Multiple language family support for programmable network systems. In *Proceedings of the 5th Annual International Working Conference on Active Networks (IWAN)*, 2003.
5. Neil Daswani and Dan Boneh. Experimenting with Electronic Commerce on the PalmPilot. *Lecture Notes in Computer Science*, 1648:1–16, 1999.
6. T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246, Internet Engineering Task Force, January 1999.
7. N. Freed and S. E. Kille. Network services monitoring MIB. RFC 2248, Internet Engineering Task Force, January 1998.
8. T. Fuhrmann, M. Schöller, C. Schmidt, and M. Zitterbart. A Node Evaluation Mechanism for Service Setup in AMnet. In *Proceedings of the 13th ITG/GI-Fachtagung Kommunikation in Verteilten Systemen (KiVS'2003), Kurzbeiträge, Praxisberichte und Workshop*, 2003.
9. D. Harkins and D. Carrel. The Internet key exchange (IKE). RFC 2409, Internet Engineering Task Force, November 1998.
10. J. Kohl and C. Neuman. The kerberos network authentication service (V5). RFC 1510, Internet Engineering Task Force, September 1993.
11. M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. J. Adams. X.509 Internet public key infrastructure online certificate status protocol - OCSP. RFC 2560, Internet Engineering Task Force, June 1999.
12. Henning Schulzrinne. GIMPS: General Internet Messaging Protocol for Signaling, June 2003.