A Node Evaluation Mechanism for Service Setup in AMnet

Thomas Fuhrmann*, Marcus Schöller, Christina Schmidt, and Martina Zitterbart

Institut für Telematik
Universität Karlsruhe, Germany

Abstract. AMnet is a programmable network that aims at the flexible and rapid creation of services within an IP network. Examples for typical services include network layer enhancements e.g. for multicast and mobility, transport layer enhancements e.g. to integrate wireless LANs, and various application layer services e.g. for media transcoding and content distribution. AMnet is based on regular Linux boxes that run an execution environment (EE), a resource monitor, and a basic signaling-engine. These so-called active nodes run the services and provide support for resource-management and module-relocation. Services are created by service modules, small pieces of code, that are executed within the EE. Based on the standard netfilter mechanism of Linux, service modules have full access to the network traffic passing through the active node.
This paper describes the evaluation mechanism for service setup in AMnet. In order to determine where a service module can be started, service modules are accompanied by evaluation modules. This allows service module authors to implement various customized strategies for node-selection and service setup. Examples that are supported by the AMnet evaluation mechanism are a) service setup at a fixed position, e.g. as gateway, b) along a fixed path (with variable position along that path), c) at variable positions inside the network with preferences for certain constellations, or d) at an unspecified position, e.g. for modification of multicasted traffic. The required path information is gathered by the AMnodes present in the network. By interaction with the resource monitors of the AMnodes and the service module repository of the respective administrative domain, the AMnet evaluation also ensures overall system security and stability.

Keywords: Programmable Networks, Active Nodes, Evaluation

1 Introduction

AMnet, the *Active Multicast Network [6],* is a programmable network that aims at the flexible and rapid creation of services within an IP network. Examples for services are manifold, including (semi) reliable multicast, mobility support, media transcoding, and many others. For a more detailed overview of AMnet services see [12]. Services are created by service modules being executed on the active nodes of the AMnet system. Both, the number of nodes required for a service and their location within the network depend on the specific service (cf. figure 1):

* Corresponding author; fuhrmann@tm.uka.de

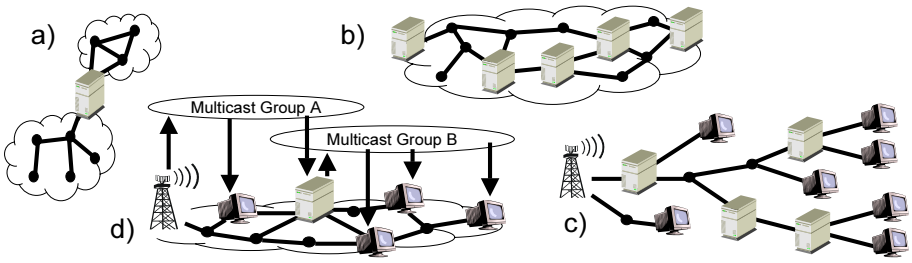

Fig. 1. The four basic location categories: (a) fixed gateway (b) on-path (c) spread on a distribution tree (d) anywhere in the intersection of two multicast groups

a) Some services require a fixed location where a service module has to be executed. E.g., a security service might be required to run on the gateway connecting the secured region with the insecure rest of the network.
b) Other services need to run at an only vaguely characterized position within the network. A media transcoding service, for example, should run somewhere on the data-path between the sender and the receiver. For a down-scaling service (e.g. PCM to MP3) this position should be close to the sender and vice versa for an up-scaling service[1].
c) For some other services the exact location (or locations respectively) where the service modules are executed can be chosen rather freely. A multicast reflector service, supporting multicast transmissions in a non-native multicast network, should ideally run at all the bifurcation points of the distribution tree. But any active node upstream from the bifurcation point may execute the service if one accepts a little bandwidth waste between that node and the actual bifurcation point.
d) In a true native-IP-multicast environment the location where a service is executed becomes even less important. A service that takes data from one multicast group, modifies it, and provides it to members of another multicast group can in principle operate anywhere as long as this position lies in the scope of both required multicast groups.

Except for the first category which defines a hard limit, all listed location requirements only yield one among several other criteria for the selection of a node. These other criteria comprise but are not limited to the following:

Memory — Can the node satisfy the service module's memory requirements without compromising the other services already running on that node?
Data rate — How many unused bandwidth is available on the node's links?
CPU cycles — Is enough CPU capacity available to execute the service module? Given that the same service might be able to run on a broad variety of different hardware

[1] Such a service is required to support end-devices that are not able to decode a certain media format.

platforms , it is challenging to compare this criterion across different nodes in the network.

Special hardware components — Service modules can require special hardware components. Presence of such components must either be treated as hard limit or - if modules use such hardware only optionally - might be treated as soft criterion that can be weighted against other criteria.

The AMnet evaluation mechanism provides a mechanism to select one[2] out of all the AMnodes that are available in the network. The selected node should satisfy all the hard criteria and match the others as good as possible, where "as good as possible" is determined by the criteria that enter the evaluation process: required evaluation time, algorithmic complexity of the evaluation, etc. In many cases, it is fair enough to contend with a heuristic instead of lengthily searching for a global optimum in the parameter space. This holds even more if one considers that the parameters considered in the evaluation process are perpetually changing, and might thus soon render each "optimal" decision sub-optimal. To reflect these constant changes in the parameter space and additionally give room for a later improvement of a quick decision, AMnet provides service modules with the possibility of relocating themselves to another, more suitable AMnode. Relocation is described in detail in [7].

The rest of this paper describes the AMnet evaluation mechanism. Section 2 gives an overview over relevant related work. Section 3 presents the basic architecture of the AMnet evaluation mechanism. Section 4 describes the relevant aspects of our implementation, and presents measurements of this implementation in the FlexiNet testbed. Section 5 concludes with an outlook on future work.

2 Related work

Following the seminal work on active networks [16] many efforts to flexibly create services inside the network focused on the capsules approach. ANTS [16], SmartPackets [13], and SwitchWare [1] directly inject code (so-called capsules) into the network, e.g. as Java bytecode or in a special purpose language [11]. This in-band approach obtains simplicity at the cost of neglecting system security. It does not require any additional management or evaluation functionality. At the same time the system has almost no control over the capsules' use of network resources[3].

With the active networking architecture [2] active packets invoke predefined functions within the network. This approach elegantly combines the straight forward code execution of the capsules approach with the additional security of programmable networks. The latter use out-of-band mechanisms to load code into the active network nodes. Other hybrid approaches are e.g. DAN [4] and the original AMnet concept [9] which used capsules only to evaluate an active node in order to determine where the actual code should be started.

[2] For readability we only consider the case where a single node executes the required service. The case of multiple active nodes is equally easily handled by AMnet.

[3] The only means of preventing capsules from consuming unlimited resources is a limited per-capsule credit that is checked by the virtual machine executing the capsule.

The evaluation problem is common to all programmable approaches since they do not have the full flexibility that capsules provide. Several possible solutions to this evaluation problem have been suggested in the literature. Darwin [8] uses a "service broker" [3] to map a resource requirement graph into actual network resources. The 2K-system [17] is based on so-called dealers that associate server, client, and active node. These dealers are organized into hierarchic domains and employ heuristics to establish appropriate associations. Other approaches [5,15] are based on the mobile agents principle [18] that determines and assigns resources in the network.

3 Architectural overview

As described in section 1, the AMnet evaluation mechanism has to select one node among all the available AMnodes and then to start the requested service there. The service request itself is either signaled automatically by an end-device or explicitly by its user. An example for the former might be a video streaming application where the video player detects severe packet loss and signals the need for a transcoding to a lower-bandwidth format. An example for the latter might be a privacy service that provides strong encryption for legacy applications that do not support encryption. There are also usage scenarios conceivable where other AMnet services signal the need for a service, e.g. a security service has detected a beginning attack and demands the installation of an appropriate filter. Still, with respect to evaluation all these cases can be treated in the same way.

3.1 Service request

A service request is signaled via a specific service request message. This message is sent towards the sender of the data which the service should operate on. Typically, a sender will be a multicast source, a (unicast) server, or a communicating peer. We can distinguish three cases of knowledge about the sender:

Case 1: Known path The service requestor knows at least one AMnode that lies on the data-path between the sender and itself. In AMnet, this is achieved by having a sender issue session announcements into which the AMnodes inscribe their IP-addresses [14]. Addresses should be enlisted cumulatively to collect a whole path of AMnodes and leave the actual choice of a node to the evaluation process. This is the easiest case since the service request can directly be sent to an AMnode where it is then handled. It requires, however, an AMnet-aware sender that regularly issues session announcements.

Case 2: Known origin If the latter is not given, we will in almost all cases still know the valid IP-address of the sender. In that case, we can reverse the session announcement mechanism described in [14] and send the service request towards the sender. If at least one AMnode lies on the data-path from the requestor to the sender, the request will be caught and the evaluation can be started. Otherwise, the sender will tacitly discard the request that it cannot handle.
This approach has two disadvantages as compared to the previous one: firstly, if no

AMnode lies on the data-path from the requestor to the sender the evaluation procedure fails or has to employ more elaborate mechanisms like they are used for the AMnet service relocation (see [7]). Secondly, with asymmetric paths this approach might find AMnodes that lie on the upstream path towards the sender but not on the downstream path from the sender to the receiver. Here, again, more elaborate mechanisms are necessary to establish an AMnet service on an appropriate node.

Case 3: Known direction In the rare case where we do not have a proper sender-address, we can still pick an arbitrary address which lies in the direction where we want the service to be put up. With that address we can revert to the previous case.

An example for this case is an attack with spoofed IP-addresses. Although these addresses are invalid, they naturally lie in the direction the attack is coming from.

The service request message that is sent according to these three cases contains a description of the requested service and identifies and authenticates[4] the requestor. Service descriptions vary in their level of strictness.

Service class level — The requested service is only abstractly specified, e.g., as transcoding class of service. The desired operation is specified by a parameter list: e.g. bandwidth limit, requested media formats (i.e. mime types), etc. The evaluation process then chooses an appropriate service module that matches the request.

Service level — The request detailedly describes the service, i.e. which kind of operation should be performed on the data. A typical example is a fixed transcoding request, e.g. PCM to MP3. Although the freedom of the evaluation process is more limited than in the previous case, the concrete choice of the service module (vendor, operating system, etc.) is still open.

Service module level — This is the most detailed case where all degrees of freedom including the actual code version are fixed. This is required since an application might rely on an unspecified feature (or even bug) of a module.

Currently, service class names, service names, and the services modules' unique identifiers are assigned administratively. At least for the latter an automated mechanism will be provided to facilitate widespread deployment of AMnet.

3.2 Evaluation Modules

Upon reception of a service request, the receiving AMnode first checks whether the requestor has sufficient rights to request that service. To this end, the node contacts one of the domain's service module repositories [14]. If the permission is granted the node receives an evaluation module that is executed to further determine the evaluation process. For services that do not require legitimation by the service requestor or for which that legitimation has already been proven and not yet expired, the evaluation module may be directly executed from the AMnode's module cache.

The code of the evaluation module has access to the relevant resource information of the local AMnode through the AMnet resource monitor [10] and to the AMnode path

[4] Required cryptographic strength of authentication depends on the service that is requested. Services may also be made generally available.

data that was signaled together with the service request. With the help of this data the evaluation module can issue a peer request to the other AMnodes on the path of the data that are to be modified. If such path information is not available, the evaluation module has to employ more sophisticated methods that are described in [7].

Upon reception of a peer request, the respective AMnode acts as previously described, i.e. it checks the legitimation of the requestor as a result of which the node obtains the evaluation module. As opposed to the initial service request, an evaluation module can also use the identity provided by the AMnode that has sent the peer request as basis for legitimation. Thus, within an administrative domain legitimation need only be checked once since peering AMnodes there trust one-another. This mechanism also provides the basis for inter-domain evaluation. Here, peering domains would mutually legitimate the use of certain evaluation modules.

The way the actual evaluation is done depends on the requirements of the service for which the evaluation is performed. Since the evaluation modules are code that is executed on the AMnodes, complete freedom can be given to the authors of an AMnet service to implement whatever evaluation suits their service module. It may just check for objective parameters as listed in section 1 or perform own benchmark tests in order to determine the node's capabilities. The outcome of these checks can reach from a plain decision whether a node can execute the service module to a fine grained judgment on how well a certain node is suited at that time for execution of the service. How these results are combined into a decision on the setup of the service is again free choice of the module's authors. Accordingly, the order and number of nodes that are contacted during evaluation also depends on the service that is to be established.

- If the service should be started as quickly as possible, the evaluation module first checks whether the local node provides enough resources to execute the actual service. The next node will only be contacted if this check fails. Otherwise the service is immediately set up. Due to AMnet's relocation mechanism an already running service can be moved to a better suited node.
- If the service preferredly runs close to the sender, the first node in the node list will be contacted and evaluated for immediate setup of the service. If that node's resources are not sufficient, the node list will be traversed reversely until a matching node has been found.
- If no location requirements are given, evaluation is performed on several nodes, either simultaneously or consecutively, depending on the evaluation module's strategy. The results of these evaluations can then be compared by the evaluation modules which then will setup the service on the node with the best result. This comparison can be done by the node initially receiving the service request, any other outstanding node, or by a distributed algorithm.

If the nodes of the node list collected in the session announcement (see section 1) is not capable of providing the service or if such a list does not exist, other mechanisms can also be deployed. These mechanisms are extensively described in [7].

3.3 Service Setup and Start

When the evaluation module(s) have decided where the service should be set up, that node is notified. It then pulls the code of the service module from the repository [14] and

Table 1. Practice Test Results

Step	Process	Average time [ms]	Percentage
1	Decode Service Request	3.3	0.4
2	Download evaluation module information (node 1)	24.3	2.8
3	Transform to Evaluation Packet	59.7	6.8
4	Download evaluation module (node 1)	36.7	4.2
5	Install evaluation module	16.1	1.6
6	Download service module information (node 1)	16.8	1.8
7	Run evaluation module	39.7	4.5
8	Send evaluation packet to node 2	95.3	10.8
9	Download evaluation module (node 2)	52.3	5.9
10	Install evaluation module	20.3	2.3
11	Download service module information (node 2)	229.4	26.0
12	Run evaluation module	46.3	5.2
13	Create and send result packet to node 1	165.0	18.7
14	Start service module	88.2	10.0
	Time until service start	881.1	100.0

starts the service. At the same time, the execution of the evaluation module terminates although its code may be kept in a cache.

In principle, the evaluation module will have checked all necessary preconditions for service setup and start. Thus almost all potential failures during setup should be treated as if an already running service fails. The only case that is treated separately are failures due to the fact that availability of node resources may change during the evaluation process. In order to avoid complex resource allocation and release mechanisms, AMnet does not reserve any resource during evaluation. If a required resource has become unavailable on the best-rated node, the setup reverts to the second-best node, etc. If this necessity to contend with seemingly sub-optimal nodes is not a rare event, resource availability is changing so quickly that service relocation is inevitable and evaluation becomes less crucial anyway.

4 Implementation and Measurements

The current implementation of an AMnet evaluation strategy is based on the known path case presented in section 3.1. The evaluation is implemented in Java using dynamic class loading and threads to integrate the functionality dynamically during runtime.

The measurements of the implementation took place in the AMnet-testbed using two AMnodes. These two nodes are both PentiumIII 800MHz standard PCs running Linux as their operating system. Both AMnodes accessed the same service module repository. The receiver was connected to *amnet1* via 100Mbit/s-Ethernet and *amnet2* was connected via a 10Mbit/s WaveLAN link. Table 1 shows the results of the practice test. The test runs resulted in an average delay of 881.1 ms from receiving a service request from a client until the service gets started. The main delay is caused by the

wireless link of node 2. To download the service module information to node 1 took 16.8 ms or 1.8% whereas the download to node 2 took 229.4 ms or 26.0%. The same applies to the communication between node 1 and node 2 in steps eight and 13. The same test runs on an all-wired testbed have reduced the average delay until service startup significantly. The evaluation of a single AMnode showed the average delay until service startup to be 354,5 ms.

5 Conclusions and Outlook

In this paper we have presented AMnet's evaluation mechanism. Its main purpose is the selection of an AMnode on which a requested service will be set up. The key idea is to use the fact that in a programmable network evaluation itself can be achieved by modules that are loaded into the active nodes on demand. AMnet thus only provides a framework in which the evaluation modules operate. Key features of this framework are basic mechanisms for discovery of the topology of the relevant AMnodes, a generic scheme to describe service requests and map their requirements onto evaluation modules, and mechanisms to identify and authenticate service requesters. The results of the evaluation modules' checks for available resources and eventually their benchmarks are combined and the service is set up on the node that seems most suitable at that time. In order to cope with changing conditions, the AMnet relocation mechanism can move already running services to more suitable nodes.

Implementation and first test of this evaluation mechanisms show that the mechanism is practical and performs well in a local setting. The focus for our future work on AMnet lies on the provision of more services employing different evaluation strategies. These evaluation approaches will then be also tested across administrative domains to show that the mechanism scales to larger networks.

References

1. D. Scott Alexander, William A. Arbaugh, Michael W. Hicks, et al. The SwitchWare Active Network Architecture. In *IEEE Network Special Issue on Active and Controllable Networks*, volume 12, pages 29–36, June 1998.
2. Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. An Architecture for Active Networking. In *INFOCOM '97*, April 1997.
3. P. Chandra, A. Fisher, C. Kosak, T. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. *IEEE Network*, 15(1):22–35, 2001.
4. Dan Decasper and Bernhard Plattner. DAN: Distributed Code Caching for Active Networks. In *Infocom '98*, San Francisco, USA, June 1998.
5. Initial active network and active node architecture. Technical report, Future Active IP Networks (FAIN) Consortium, May 2001.
6. Thomas Fuhrmann, Till Harbaum, Marcus Schöller, and Martina Zitterbart. AMnet 2.0: An Improved Architecture for Programmable Networks. To appear in: Proceedings of IWAN'02, available from http://www.flexinet.de.
7. Thomas Fuhrmann, Marcus Schöller, Uwe Freese, and Martina Zitterbart. Service Relocation in AMnet. Available from http://www.flexinet.de.

8. 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.
9. Till Harbaum, Anke Speer, Ralph Wittmann, and Martina Zitterbart. Providing Heterogeneous Multicast Services with AMnet. *Journal of Communications and Networks*, 3(1), March 2001.
10. Andreas Hess, Marcus Schöller, Günther Schäfer, Adam Wolisz, and Martina Zitterbart. A dynamic and flexible Access Control and Resource Monitoring Mechanism for Active Nodes. In *Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH) (Short Paper Session)*, 2002.
11. Michael W. Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A Packet Language for Active Networks. In *International Conference on Functional Programming*, pages 86–93, 1998.
12. The FlexiNet Project. http://www.flexinet.de.
13. Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, R. Dennis Rockwell, and Craig Partridge. Smart Packets: Applying Active Networks to Network Management. *ACM Transactions on Computer Systems*, 18(1):67–88, 2000.
14. Anke Speer, Marcus Schöller, Thomas Fuhrmann, and Martina Zitterbart. Aspects of AMnet Signalling. In *Networking 2002*, pages 1214 – 1220. Springer, March 2002.
15. A. Tan and A. Galis. Active IP Network Node Developments. Technical report, Department of Electrical & Electronic Engineering, University College London, 2000.
16. David J. Wetherall, John V. Guttag, and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH '98*, San Francisco, USA, April 1998.
17. Dongyan Xu, Klara Nahrstedt, and Duangdao Wichadakul. MeGaDiP: A Wide-Area Media Gateway Discovery Protocol. In *Proceedings of IEEE IPCCC 2000*, February 2000.
18. Yechiam Yemini and Sushil da Silva. Towards programmable networks. In *Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italien, October 1996.