RZ 3437 (# 93625) 07/15/02
Computer Science 12 pages

Research Report

Adaptive End-to-End Quality-of-Service Guarantees in IP Networks using an Active Networking Approach

Roman Pletka

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland
rap@zurich.ibm.com

Burkhard Stiller

ETH Zürich
Computer Engineering and Networks Laboratory (TIK)
8092 Zurich
Switzerland
stiller@tik.ee.ethz.ch

Adaptive End-to-End Quality-of-Service Guarantees in IP Networks using an Active Networking Approach

Roman Pletka[1] and Burkhard Stiller[2]

[1] IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland,
`rap@zurich.ibm.com`,
[2] ETH Zürich, Computer Engineering and Networks Laboratory (TIK), 8092 Zürich, Switzerland,
`stiller@tik.ee.ethz.ch`

Abstract. This paper proposes a framework based on an active networking approach to efficiently link Quality-of-Service (QoS) descriptions from an application point of view with an underlying heterogeneous IP networking infrastructure. The main goal is to perceive the availability of and deploy distinct QoS capabilities in order to accomplish adaptive end-to-end service guarantees. The building blocks needed in a heterogeneous IP network will be introduced and discussed with respect to safety in terms of total networking bandwidth, CPU, and memory usage. In particular, the problem of QoS parameter translation to provide adaptive end-to-end service guarantees is addressed and an example using Diffserv and RSVP in a heterogeneous network is given.

1 Introduction

The demand in real-time applications and the increasing number of multimedia traffic streams cannot be satisfied using a best-effort-based network. Service differentiation becomes a must for future IP networks. In the past, several QoS architectures have been proposed and standardized in the IETF [1, 2]. Nevertheless, QoS support is rarely used in heterogenous IP networks because end-to-end support of service guarantees lacks. Furthermore, the increasing variety in QoS provisioning mechanisms (e.g., schedulers and active queue management) in network nodes complicates their integration into QoS frameworks, and the abstraction of QoS descriptions make static assignments between different QoS frameworks impractical.

QoS support can be given as absolute guarantees using a signaling mechanism as in the Intserv/RSVP [1] approach or in a differentiated manner on top of abstract and simple building blocks as provided by Diffserv [2]. Full end-to-end service using Intserv is only feasible if the negotiation process is successful – this means that all intermediate hops need to support the QoS requirements throughout the lifetime of the corresponding flow.

Packets entering a Diffserv network are classified based on contracted parameters given in Service Level Agreements (SLA) according to their service requirements, and are marked to receive a particular per-hop behavior (PHB). The required functionalities at Diffserv domain edges are classification and conditioning (including policing and shaping) in accordance with the policies at the edge. High scalability is achieved by applying PHBs to aggregates of similar traffic. Diffserv building blocks do not describe the implementation of the underlying hardware capabilites and functionalities that describe PHBs. Although Diffserv is highly scalable, it has no inherent support of quantitative QoS. Moreover, Diffserv alone is not sufficient for providing end-to-end guarantees, as data flows may cross several Diffserv domains.

Bernet *et al.* [3] outline a model for a complementary inter-operation between Diffserv and Intserv where Diffserv is used by transit networks in the core of the Internet while hosts and edge routers use RSVP/Intserv. The framework suggests that end-to-end, quantitative QoS can be provided by applying the Intserv model end-to-end across a network containing one or more Diffserv regions. These Diffserv regions can be seen as network elements in the end-to-end path. The mapping from Intserv to an appropriate Diffserv Codepoint (DSCP) may be hard coded per default, dynamic, or customer specific. Although there is a rich variety of possible approaches to perform admission control and the corresponding service mapping, this task is extremely difficult and, besides depending heavily on the network topology and the current status of the network load, entails the use of bandwidth brokers [4] between Diffserv regions.

This paper presents a framework for adaptive end-to-end QoS guarantees using active networks to optimize the usage of existing QoS capabilities. The building blocks required in a heterogeneous Internetwork will be introduced and discussed. Active packets are used for QoS provisioning in conjunction with and as a complement to existing frameworks. Our framework allows the translation of QoS parameters from an unsupported scheme into one that is supported by the networking infrastructure considered. This can be done for a particular router or a heterogeneous domain, and in horizontal (e.g., between networking nodes) as well as vertical (e.g., inside a given networking node) direction. The technique uses a stream code active networking approach. Safety is guaranteed by two different means: First, by restrictions in the active byte-code itself, and second, through the definition of a safety hierarchy. The safety hierarchy allows code to be placed in active routers only under certain restricted conditions.

The framework allows service providers to dynamically define and install QoS translation services, which are necessary owing to the heterogeneity of the network. Furthermore, it allows applications and networking nodes to describe their service requirements by placing active code into packets. This code is then used within the network to facilitate the translation and adaptation process in order to find an appropriate service behavior.

The remainder of this paper is structured as follows. Based on a brief survey of major related work on end-to-end QoS support and active network frameworks in Section 2, the proactive QoS environment is presented in Section 3. The requirements and implementation guidelines for the active networking approach are given in Section 4, followed by an example (Section 5) to gain a better insight into the new framework. Finally, Section 6 presents our conclusions.

2 Related Work

In [5], Guérin and Peris point out the challenges and trade-offs for interoperability of different QoS mechanisms. They provide certain requirements for backbone networks that need to be fulfilled in order to enable scalable end-to-end QoS guarantees. The solutions envisaged that fulfil these requirements can be categorized as static, i.e., relying on provisioning of different service classes on backbone links, or dynamic, i.e., allowing to adjust the level of aggregated reservations. Static solutions have obvious limitations in how they handle changes in the network (e.g., delay variations due to route changes). The main challenge of dynamic solutions lies in properly specifying aggregated reservations while preserving per-flow fairness.

The *Smart Packets* approach for active networking introduced by Schwartz *et al.* [6] focuses on network management and monitoring. To simplify management and consistency of active code, routers do not maintain state across packets.

The packet-transport service is connectionless and smart packets must be self-contained. Therefore, programs have to be smaller than 1 KB. Security concerns restrict the active code to be executed within a sandbox environment. A C-like language called Sprocket is compiled into a machine-independent assembler, which in turn is assembled into byte-code for a virtual machine. Execution is conservative in the sense that an error message is sent back if the virtual machine does not know how to handle the operation. Extending services dynamically is not feasible with Sprocket as it would require modifying the virtual machine itself. The security architecture consists of a maximum number of instructions to be executed, limited memory usage, and restricted access to the management information base.

Moore *et al.* [7] balance the tradeoffs between flexibility, efficiency, and safety with SNAP (Safe Networking with Active Packets). SNAP, a stack-based active networking language, evolved from PLAN (Packet Language for Active Networks) and is safe with respect to network resource usage (resource conservation) and evaluation isolation. The execution of a SNAP program can only consume bandwidth, CPU, and memory resources up to a limit that is linear to the packet length. To achieve this goal, only forward branches are allowed, i.e., loops and recursions are prohibited. Furthermore, packets may only stay at nodes for a limited amount of time. This balance of features distinguishes SNAP from other active networking approaches that are either restricted to the control plane, have unacceptably low performance, or sacrifice safety.

In [8], Keller *et al.* present an active router architecture for multicast video distribution and show that video quality can be improved by adapting the video stream on routers using router plugins. Router plugins are retrieved from a code server and installed in the router's kernel before the video distribution. A video scaling algorithm implemented as an active router plugin scales the video on-the-fly to the appropriate target bandwidth determined by the actual networking load. Owing to the implementation as a kernel module the router plugin is able to perform scaling extremely fast, namely, on average in 22 μs for a datagram.

A hierarchical mechanism for scalable service deployment has been introduced in [9]. The proposed framework allows the distributed and dynamic capabilites in a heterogeneous network to be captured and organizes the deployment based on specific service policies in a programmable manner. Five steps form the core of the mechnism. Solicitation identifies the required information on capabilities in the nodes. Summarization packs the metrics for scalability reasons. Dissemination allows a particular service to be distributed from the top-most node to all physical nodes involved. Then, installation takes place independently in the physical nodes before advertisement is used to guide packets that rely on this particular service through the network to the destination.

3 QoS Provisioning in a Heterogeneous Environment

QoS guarantees are required for a set of important applications, which operate across sub-networks and multiple domains. Therefore, QoS provisioning in a heterogeneous environment, which offers those functionalities and enables a mapping from a service request to the networking parameters in an end-to-end fashion, is required.

3.1 Overview

A QoS request from an application implies the steps shown in Figure 1. Note that some of them might be rather simple and point to default best-effort policies if no other services are registered.

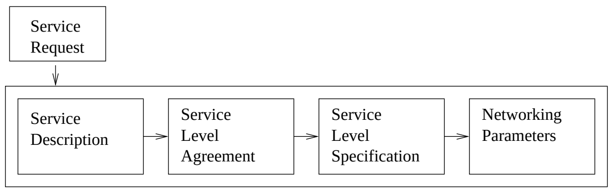


Fig. 1. *Quality-of-Service parameter sequence.*

Service description: End users rarely know exactly what kind of service guarantees they need for a certain application. Therefore most of the time the application itself knows the suitable service description. As an example, a rate of 8 kb/s (G.729) and a maximum end-to-end delay of 400 ms may describe a Voice-over-IP (VoIP) service. The application initiates a QoS-demanding data flow by setting up a RSVP reservation, for example by simply marking data packets with a certain DSCP, or by inserting active code into data packets.

Service Level Agreement (SLA): An SLA is a contract between a customer and a service provider and defines a network boundary relationship. It specifies the forwarding services a customer should receive and may include traffic conditioning rules. SLAs also define the provider's responsibilities expressed as QoS parameters (e.g., throughput, loss rate, delay, and jitter), availability, measurement methods, consequences when the defined service cannot be met, and the costs associated to the service. An enduser should not be involved in SLA description or SLA negotiation directly.
Service Level Specification (SLS): An SLS contains the technical characteristics for an SLA, and describes the resources provisioned for this particular service. There is one SLS for each service specified in an SLA. An SLS neither defines implementation details nor does it give indications on the networking resources available beyond the boundary concerned.
Networking Parameters: Finally, SLSs will end up in networking parameters that consist of hardware parameters (for buffer management, scheduling algorithm and active queue management), domain policies, and end-to-end flow control mechanisms. In a heterogeneous network, hardware support can differ for each router. Networking parameters are neither reflected in SLSs and SLAs nor in the description of the service, as they describe the boundary at the edge of the network rather than the behavior of nodes in the network. Static pre-configured networking parameters lead to poorly exploited existing capabilities from the outset. Note that in our approach the routing process itself is not strictly a networking parameter, because the path a packet takes through the network is maintained by routing protocols – unless it is an alternative route. Thus, routes cannot be modified without interfering with routing protocols as this would lead to inconsistencies in the traffic engineering information maintained by the routing protocol stacks, and might result in completely unpredictable networking behavior.

The difficulty of maintaining the QoS parameter sequence for end-to-end services stems from the missing knowledge of the actual status of the network beyond the network edge, and the need of a per-flow inter-domain bandwidth broker raises scalability issues.

3.2 Proactive QoS Environment

From a traditional network administrator's view, network availability can be detected using the Simple Network Management Protocol SNMP [10] to monitor corresponding managed objects contained in the Management Information Base (MIB). Although network management traffic has in general the highest priority, the information has to traverse the network before it can be evaluated by the network manager or an appropriate network-management tool. Moreover, the use of SNMP entails additional inbound traffic that imposes limitations on the frequency at which information can be updated.

In order to apply the necessary QoS translation, QoS information is kept locally and exchanged between adjacent nodes, which allows the QoS information to be brought to where it is needed. Dynamic QoS parameter configuration increases responsiveness and adaptability while reducing inbound traffic. We propose the following two-level technique:

– *Intrinsic QoS adaptation*: Based on service descriptions and QoS translation in nodes, active code placed in packets facilitates queuing and dropping decisions. This is, of course, done within certain well-defined bounds that are controlled on-the-fly in the network and are supplied by an end-to-end control mechanism. If a packet does not conform and thus exceed these bounds, it will not be able to traverse the network and accomplish its task.[3] The network will discard such a packet without further investigation. It is clear that packets are subject to certain time constraints, which usually are imposed by the application level, and that there is a tradeoff between functionality and additional delay.
– *End-to-end control mechanism*: The global behavior of a packet stream can be optimized using a traffic profile that includes the QoS description. Such a traffic profile can be related to the application's needs but could also be the result of a policing mechanism at the network edge or the translation of QoS parameters from an unsupported scheme into one provided by the current networking domain. The attitude of a packet is optimized by the definition and adaptation of packet's behavior bounds, which are a part of the QoS description, according to the current status of the network. Active code describes the behavior bounds which are carried in an active packet.

Figure 2 illustrates the proactive QoS environment that consists of three different operating planes. The application plane describes and requests specific QoS guarantees from the network, which will be provided to an application. QoS signaling can be implicit or explicit, and the QoS description absolute or relative within certain statistical bounds. Examples are Intserv, Diffserv, or active packets that contain this information in the form of small active code sections.

[3] The word task is used to distinguish between the traditional point of view where a packet consists of header and payload, and the active network approach where packets contain some sort of byte-code that is executed in the network and therefore fulfills a certain task.

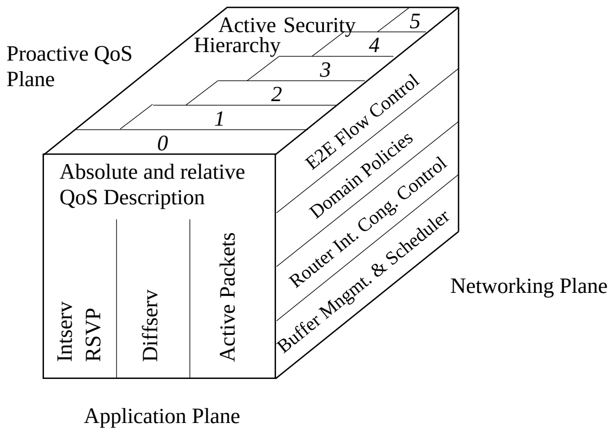


Fig. 2. *QoS provisioning in a proactive environment.*

The networking plane consists of per-hop-based networking parameters, domain policies, and end-to-end flow control mechanisms. This plane reflects the mapping of QoS descriptions into hardware functions placed at the disposal of the corresponding node.

Finally the proactive QoS plane allows the translation of QoS parameters from the application plane to the networking plane, and acts according to these parameters on a per-hop basis. To fulfill this task, the proactive QoS plane utilizes and maintains behavior bounds. Given the heterogeneity of networks in terms of underlying hardware as well as domain-specific behaviors, the proactive QoS plane cannot be described by simple static means, e.g., a single end-to-end connection already needs a multitude of description blocks. The functionality of the translation process is limited by six safety levels (cf. Section 4.4) numbered from zero to five. Functionalities provided by low-numbered safety levels can be used in the data-plane packet-forwarding process, whereas higher-level functionalities include operations on policies and router services, which in general require more resources and are subject to restrictive policies. Eventually, an end-to-end service requiring a certain QoS guarantee can be achieved by chaining these description blocks as shown in Figure 3.

From this approach the following implicit functionalities of a networking packet can be derived: Packets are no longer static data containers but can also carry active program code. Data and program code can reside in packets at the same time. These functionalities enable dynamic QoS parameter translation between different QoS frameworks (e.g., Diffserv and Intserv) that are not translatable by simple one-to-one mapping functions.

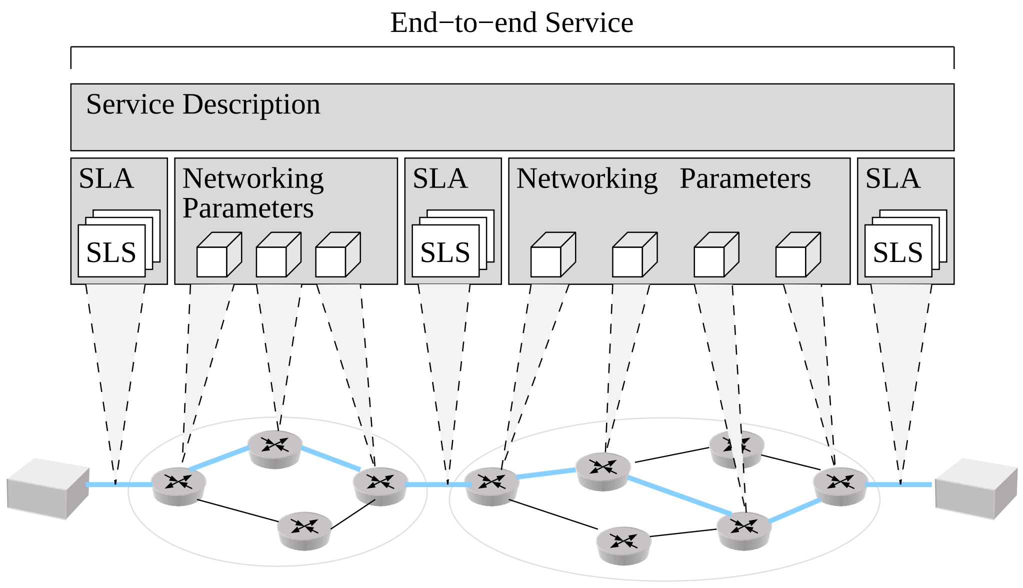


Fig. 3. *End-to-end services in a proactive QoS environment.*

3.3 Functional Description

Before we show how active packets are used to program the proactive environment, we introduce the functional modules necessary in an active router. They consist of four different building blocks: the discovery process, the QoS translation phase, resource management, and feedback mechanisms.
Discovery process: The discovery process builds initial behavior bounds that describe upper bounds of service limits corresponding to maximum available resources, in order to obtain some a-priori knowledge of QoS availability prior to connection setup. The information gathered on instantaneous resources is then updated periodically. A possible source of information is the traffic-engineering opaque LSA messages in OSPF [11]. Unlike the QoS Broker [12], the discovery process is used within the network and does not deal with QoS discovery from the application or enduser's view point. The end-to-end QoS behavior is therefore subject to an ongoing and adaptive process throughout the lifetime of a connection, and is a result of the network discovery process present in the background.
Translation phase: The necessity of active code stems from the fact that QoS translation in general is not a bijective operation, and therefore increases complexity. The following guidelines can be used: Surjective code translation is obtained by projection onto the new QoS space, whereas injective code translation needs additional information based on default mappings and/or educated-guess methods. Bijective translation is primarily achieved with one-to-one table-based mapping. The translation process is done using active code provided by either the network administrator or in certain cases by the application itself as long as safety is not compromised. Classification of the packet allows the appropriate safety level and, if needed, an adequate translation code to be chosen.
Resource management: Resource management comprises the task of maintaining information on the actual status of resource availability. A certain share of the resources can be initially assigned to the active networking element. The resources that are administered consist of QoS-related resources (e.g., maximum bandwidth per traffic class), policies, resources related to the neighborhood, and router services. From these resources the behavior bounds are derived.
Feedback mechanisms: Instantaneous traffic characteristics can deviate from the corresponding QoS reservation and are influenced by numerous factors in the network (e.g., traffic shapers, actions of active queue management (AQM) schemes, the granularity of schedulers, amount of cross-traffic), and in end systems (e.g., round-trip time in TCP). All these factors are variable in time, and affect the end-to-end service. Some of them are controlled by specific feedback mechanisms. Adaptive end-to-end service guarantees are feasible when the interaction between feedback mechanisms is taken into account i.e., their interactive behavior is predictable. The proactive QoS environment uses feedback mechanisms that act in an active node as well as between neighbors, and uses different time scales to update behavior bounds.

4 Active Networking Framework

There are several reasons why an active networking approach should be deployed. The fast increase of the backbone traffic volume in the Internet and the introduction of QoS support overextend the management tasks, and passive network management simply does not scale. Moreover, the successful introduction of any new protocol must be guided by a standards body and is in general a slow process. Although a protocol can be very rich in functionalities, it is limited by a well-defined function space. Inter-operation between different protocols, as can be encountered in heterogenous environments, is beyond the scope of protocols. In addition, no translation mechanisms between existing QoS frameworks are provided. Adding new functionalities in a network implies the complex task of modifying and adding protocols in routers.

Active networks have the advantage that new functionalities, in our case focused on QoS support, can be deployed dynamically. They offer the functionalities necessary for a correct mapping of service descriptions to networking parameters. To some extent active networks increase the flexibility in QoS-enabled networks.

One of the major concerns with active networks is security. Our approach relies on safety in terms of total networking bandwidth, CPU, and memory usage, a general safety hierarchy, and a sandbox environment for code execution in order to reduce security risks to the level of traditional IP networks in which a variety of existing solutions can be found. Networking security aspects would exceed the scope of this contribution. Below we first discuss the requirements for an active networking approach and then show how these requirements can be met.

4.1 Requirements

Based on the insight gained in Section 3 let us outline the main requirements for an active networking approach:
Byte-code language: Architectural neutrality is achieved with a byte-code representation in conjunction with an execution environment to execute the program code provided securely. The byte-code language has to support basic functionalities, including standard arithmetic and relational operations, and must provide exact safety properties for CPU and

memory usage. It should not be focused on any kind of application. A simple byte-code language allows immediate and dynamic QoS adaptation as well as the installation and maintenance of higher-level router services.
Resource bound: Network resources are given by a two-dimensional vector that consists of a local part, which is consumed on a router while executing byte-code instructions, and a network part which limits the spread of a packet.
Safety levels: The handling of active networking packets is split into six security levels. Admission control and accounting is done at the edge of the network and depends on the Internet service provider's (ISP) needs. Thus active packets from outside a domain that are accepted can be counted for charging purposes, whereas higher-level packets can be filtered.
Sandbox environment: Any active byte-code is executed in a safe environment called the Active Networking Sandbox (ANSB). Information exchange with the router is only feasible using router services.
Router services: The limited functionality of byte-code instruction can be dynamically extended by using router services. Some frequently used router services (e.g., IP address lookup, creation of a new active packet) that represent common networking tasks are present in the byte-code language definition and are therefore static. The difference between byte-code instructions and router services is justified by the significantly higher resource usage of the latter. Protection against misuse is given by the different safety levels in the safety hierarchy.
Routing: Active packets will not interfere with routing protocols. Alternative routes can be proposed by router services as long as the corresponding entries are defined in the local routing table.

4.2 Byte-Code Language

The active networking framework following the stream code active networking approach uses an extension of the SNAP [7] instruction set as shown in Table 1. The properties of this stack-based language – SNAP is a byte-code language providing safety with respect to network resource usage and evaluation isolation – fit best the requirements mentioned above and justify the choice. Core service instructions have been added to cover frequently used router services, such as accessing the interfaces, queuing classes, and congestion status. Other changes have been made to simplify heap operations (`STO` and `RCL`). A frequently used stack operation (`SWAP`) has been added. `GETLRB` delivers the local resource bound, as will be explained later. `GETENTRY` and `SETENTRY` allow the manipulation of the ingress and egress entry points.

Heap and stack elements are 32 bits in size and instructions are encoded in 16 bits; seven for the opcode and nine for immediate arguments. To simplify access to active packet data, the memory section is located between the packet header and the active code section (see Figure 4).

Table 1. Instruction classes.

Instruction Class	**SNAP**	**SNAP Dialect**
Stack control	`EXIT, PUSH, POP, POPI, PULL`	`SWAP`
Flow control	`PAJ, JI, BEZ, BNE`	Backward branches allowed
Heap operations	`MKTUP, NTH`	Replaced by `RCL, STO`
Relational operators	`EQ, EQI, NEQ, NEQI, GT, LT, GEQ, LEQ`	
Arithmetic operators	`ADD, ADDI, SUB, MOD, NEG, NOT, AND, OR, ORI, XOR, LSHL, RSHL`	
Operation on addresses	`SNET, BCAST`	
Static router services	`GETSRC, GETDST, FORW, SEND, HERE, ISHERE, GETRB, ISNEIGH`	`GETITF, GETNQ, GETENTRY, SETENTRY, GETCS, GETLRB`
Dynamic router services	`CALL, RREQ`	

4.3 Resource Bound

To prevent a denial-of-service attack, SNAP uses several restrictions to limit resource utilization of active packets in the network. These restrictions consist of a resource bound derived from the time-to-live (TTL) field and the fact that SNAP programs use bandwidth, CPU, and memory resources in linear proportion to the packet's length [13]. Hence, byte-code instructions must execute in a constant and predictable time frame.

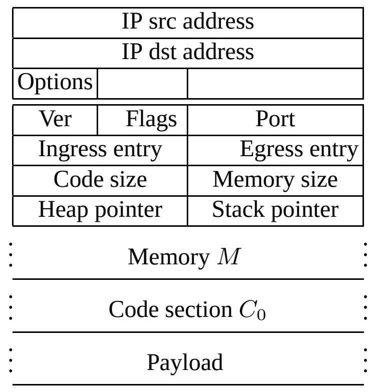


Fig. 4. *Active packet including parts of the IP header. Ingress and egress entry point to the corresponding starting points in the code section C_0. Heap and stack pointer indicate the current positions in the memory section M.*

The resource bound as defined in SNAP turns out to be difficult to respect because byte-code instructions may differ significantly in terms of their execution time [14]. In addition, the limitation that only forward branches are allowed is too restrictive for real programs. Therefore, a new definition of the resource bound is introduced here.

Consider two packets with the same packet length. One has a large code section and no payload, the other has a large memory and payload section. The packet with the large code section should receive the same resources for executing active code as the packet with a large memory and payload section. In other words, as processing a packet p takes at most $O(|p|)$ time ($|p|$ denotes the packet length), the payload and memory section can be accounted for the packet's resource budget in the same way as code sections. The idea is to use this extra budget for program loops.

We define a packet p as $p = \langle r, C_0, C, M\rangle$ using the notation borrowed from [13], where r is the network part of the resource bound, C_0 the full program code, C an arbitrary sequence of instructions within C_0, and M the memory size containing heap and stack of the packet.[4] The notation $\longrightarrow_{local}$ describes the execution of an active packet on a given node and $\longrightarrow_{net}$ the behavior of a packet in the network; $\longrightarrow^{j}$ is a sequence of j reductions. $|C|$ denotes the number of dynamically executed byte-code instructions of the sequence C. The SNAP reduction for CPU safety

$$\langle r, C_0, C, M\rangle \longrightarrow^{j}_{local} \langle r, C_0, \emptyset, M'\rangle \quad j \le |C_0|, \tag{1}$$

no longer holds in the presence of loops because $|C|$ might be larger than $|C_0|$. Moreover, byte-code instructions do not execute in constant time. Therefore, we define $\hat{C}$ as the sequence of machine-dependent CPU cycles corresponding to the byte-code sequence C, and denote $|\hat{C}|$ as the number of machine-dependent CPU cycles thereof.

In the following, we show that our extension of the SNAP resource concept does not violate safety on CPU, memory, and bandwidth usage. The new resource bound, a two-dimensional vector[5], consists of a local part $n|p|$, which is linear to the packet size $|p|$, and a network part r proportional to the TTL of the packet (n is a constant of proportionality measured in instructions per byte). While the conditions on the network part remain unchanged, the resource bound for the local part has to be redefined. The new definition of the local resource bound is based on maximum instruction costs w_i (derived from the real execution times) associated with each instruction i in the byte-code instruction set I.

For an active packet with data D (where D is the entire packet minus the code section C_0), the local resource bound is proportional to the sum of the sizes of code section $|C_0|$ and data $|D|$. This sum has to be greater than the sum of all maximum instruction costs w_i (in native cycles for the byte-code $i \in I$ independent of any program) executed for the packet in total. Thus, CPU safety holds under the following condition

$$n|p| = n|C_0| + n|D| \ge n|C| = m|\hat{C}| = m\sum_{k=1}^{|C|} w_{C[k]} , \tag{2}$$

where $C[k]$ is the k-th instruction in the execution sequence. Note that $C[k] \in I \quad \forall k : 1 \le k \le |C|$. The factor m is determined by the average instruction cost and the maximum number of processor cycles the ANSB can spend on executing active code for a given packet.

[4] For simplicity the notation of a packet does not contain all parts of a packet (e.g., packet header is not represented). Only the relevant parts are included.

[5] The resource bound is defined as a vector because local and network resources are non-exchangeable for safety reasons.

We are now going to prove Equation (2). Let $|\hat{i}_b|$ be the number of native (single-cycle) instructions needed to interpret byte-code $i \in I$ given a machine state such that the interpreter will take branch b out of the set of all possible branches B_i for that opcode. During initialization time, all w_i have been chosen to satisfy the following equation,

$$\max_{b \in B_i} |\hat{i}_b| \leq w_i \qquad \forall i \in I \tag{3}$$

by taking the longest branch possible. $|\hat{C}[k|b]|$ is the number of native instructions in the sequence $\hat{C}[k]$ given that the machine state is such that the interpreter will take branch b out of the set of possible branches $B_{C[k]}$. Equation (3) holds for every instruction i defined in the byte-code language and is independent of any code sequence C:

$$\max_{b \in B_{C[k]}} |\hat{C}[k|b]| \leq w_{C[k]} \qquad \forall k : 1 \leq k \leq |C| \quad . \tag{4}$$

By induction on $\longrightarrow_{local}$ each instruction i decreases the available local resources by w_i. Therefore the sum over an arbitrary sequence of instructions is limited by the resource bound given in Equation (2):

$$m \cdot \sum_{k=1}^{|C|} \max_{b \in B_{C[k]}} \left| \hat{C}[\,k \,|\, b\,] \right| \leq m \cdot \sum_{k=1}^{|C|} w_{C[k]} \leq n|p| \tag{5}$$

$\square$

The less restrictive bound on memory usage defined in SNAP remains unchanged for the following two reasons: First, all byte-code instructions can push at most one element on the stack or add at most one element to the heap. Second, using (2), the maximum number of instructions that can be executed is bounded and the maximum growth of the stack does not depend on the presence of loops.

The detailed proof is similar to the one given in [13]: Introducing the primitive operator $op_{p,q}[k]$, with p stack values inputs and q stack values outputs for the instruction k, a growth function can be defined as $growth(op_{p,q}[k]) = q - p$. Each local reduction adds at most $m = \max_{p,q,k}(growth(op_{p,q}[k]))$. Thus, the growth of the stack is limited by a multiple m of the executed byte-code instructions $|C|$. Using Eq. (2), the stack growth is bounded by the local resource bound which is proportional to the packet length. $\square$

Finally, bandwidth safety and processing isolation remain unchanged. The former is given by the fact that only the local resource bound has been modified, and the latter can be shown as follows: Given two packets p_1 and p_2, an arbitrary number of reductions in the network ($\longrightarrow^*_{net}$) create N_{p1} and N_{p2} packets. Thus, $p_1 \longrightarrow^*_{net} N_{p1}$ and $p_2 \longrightarrow^*_{net} N_{p2}$. Processing isolation is given when $p_1 \cup p_2 \longrightarrow^*_{net} N_{p1} \cup N_{p2}$. As the $\longrightarrow_{net}$ reduction affects one packet at a time,

$$p_1 \cup p_2 \longrightarrow^*_{net} N_{p1} \cup p_2 \longrightarrow^*_{net} N_{p1} \cup N_{p2} \quad . \tag{6}$$

Thus, memory safety as well as processing isolation remain unchanged. $\square$

4.4 Safety Levels

In this section an adequate solution for scalable and safe active networking, the safety levels, are introduced and discussed. The goal is to protect networking resources from malicious users and to distribute excess resources fairly at the same time. It is not wise to overwhelm the network with strong cryptographic measures with regard to the data plane in which each packet has to be authenticated and encrypted. The network simply has not enough resources to handle these packets in the data path. Therefore, an adequate and sufficient safety hierarchy is a mandatory building block.

Packet classification (a policy-based procedure) determines the safety level of packets. Higher safety levels are likely to be coupled with more restrictive policies. Thus, the safety levels of the proactive QoS plane have a pyramidal shape, which restricts the execution of powerful commands according to policies, and therefore achieves the required safety.

Table 2 shows the proposed safety hierarchy that addresses the problem. Levels 0 and 1 of the hierarchy address the data path. Level 0 corresponds to traditional packet-forwarding process without execution of active code. Packets containing simple active byte-code are allowed on level 1. Safety is solved by restrictions in the language definition and the use of a sandbox environment. SNAP [7] is an example of such a byte-code language. A simple packet byte-code enables immediate QoS provisioning and minimizes security efforts because no verification of plugins has to be done. In traditional routers, QoS decisions are taken using local information and information from packet headers. The main advantage of the new approach is that packets are able to take additional information from preceding hops into account, hence acting in a flexible and distributed manner. A router will execute the active byte-code in a special sandbox environment that ensures safety.

Level 2 provides certain router-service primitives in addition to the byte-code language. As router services in general are more costly in terms of CPU cycles than simple byte-code instructions, and can differ significantly in the cost as

Table 2. Safety hierarchy in active networks.

5	Dynamic services: registering new router services	Authentication of active packets needed (i.e. Proof carrying code and public key infrastructure).
4	Complex policy insertion and manipulation	Admission control at the edge of the network, trusted within a domain.
3	Simple policy modification and manipulation	Running in a sandbox environment, limited by predefined rules and installed router services.
2	Creation of new packets and resource-intensive router services (lookups etc.)	Sandbox environment based on the knowledge of the instruction performance.
1	Simple packet byte-code	Safety issues solved by restrictions in the language definition and the use of a sandbox.
0	No active code present in packets	Corresponds to the traditional packet forwarding process.

well, the definition of a resource bound as given above is mandatory. Router services allow packets to obtain information on congestion status and policies installed at the given hop.

The next higher level, i.e., level 3 allows modification and manipulation of policies that are installed using router services. As an example the RSVP soft-state mechanism is mentioned here in which state information is updated by RSVP reservation request (Resv) messages to maintain reservation state and RSVP Path messages that store path state in each node along the way. Unlike RSVP this level does not allow the installation of new states in routers.

The insertion of policies and complex rule manipulation is handled by level 4. In the RSVP example, this reflects in the creation of new state information in a router.

Level 5 finally allows the installation of dynamic router services that can provide and maintain information for active packets in lower levels. For security reasons, no level 4 packets from outside a service provider's network are allowed to enter the domain. Thus only certain management hosts within the ISP domain are allowed to inject such packets. Nevertheless when a certain well-known service should be installed, agents that reside at the network edge can, for example, translate the requested service into a predefined level 4 active code that will then install the service in the ISP's network.

4.5 Sandbox Environment

Figure 5 gives an overview of the Active Networking Sandbox (ANSB) and shows its modules in the data and control planes. The active networking framework supports service primitives that can be requested from the ANSB. This is similar to the plug-in approach [15], but is used here to offer customized networking services that are time-constrained and do not allow the execution of arbitrary code segments (e.g., IP address lookup, QoS parameter lookup).

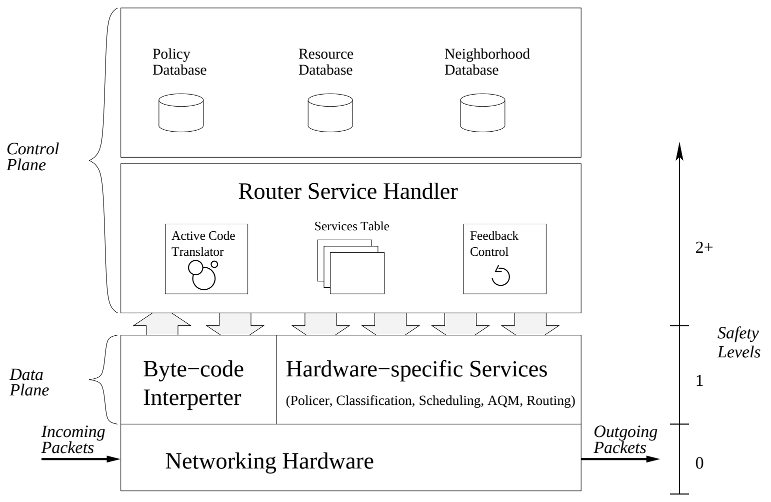


Fig. 5. *Active networking sandbox.*

The ANSB provides safe execution of active code because memory access and resource utilization is strictly controlled by the resource bound. For example, full read/write access is only granted for the memory section of the packet, whereas read access is only given for the packet header. Other data structures are only accessible using router services. Thus active network code is not allowed to write data at any location in the packet or ANSB environment.

Policy information is stored in a policy database. The resource database keeps track of the locally available resources. The neighborhood database maintains information such as maximum bandwidth and maximum available bandwidth of its neighbors. A source of information for the neighborhood database could be the opaque LSA messages from the OSPF routing protocol [11]. The active code translator manages dynamic router services (register/deregister) and dispatches requests for installed router services from the byte-code interpreter.

The ANSB has two separate tasks. The first is to control the safe execution of active code of an incoming active packet. The second is a background task to maintain tables. An example of such a table is the congestion-status table which keeps track of the occupancy of the queues on a given output interface. Depending on the underlying hardware, the ANSB can run in the control point (e.g., on a Linux router) or directly in a network processor.

4.6 Router Services

Router services are functions that go beyond the definition of byte-code instructions. They can be static, i.e., defined in the byte-code language (e.g., IP address lookup, getting the number of interfaces, flow queues, or congestion status information), or dynamic (e.g., installation of new active code into the ANSB service table for QoS translation, policy manipulation using a dynamically loaded router service), when installed on nodes. Usually, they are tailored to networking tasks with focus on control-plane functionalities, and execute in a significant higher amount of time than normal byte-code instructions. Therefore, router services belong to the set of instructions with a safety level higher than 1. Furthermore, the installation of new router services is restricted to safety level 5. Such an active packet contains the context for which the new service is applicable, and the code section that will be installed is given in the packet's payload. Analogously, dynamic router services and their context information can be removed, modified, or updated.

Insertion, manipulation, and removal of policies is feasible within a given context. Simple static router services on safety level 2 provide information on queues, their congestion status, routes, or context-based lookups. The next section gives more detailed insight into the functionality of router services by means of an example.

5 Example Applications

As an example, a controlled load RSVP service using a fixed filter is discussed (cf. Figure 6). The service is provided over a heterogeneous network that can be split into three different domains.

Domain 1 supports full RSVP on all routers in the domain and might correspond to a small ISP placed at the edge of the network (e.g., in the metropolitan area) and therefore is capable of handling per-flow RSVP signaling. This domain does not provide active routers at all. Active packets are forwarded as regular IP packets.

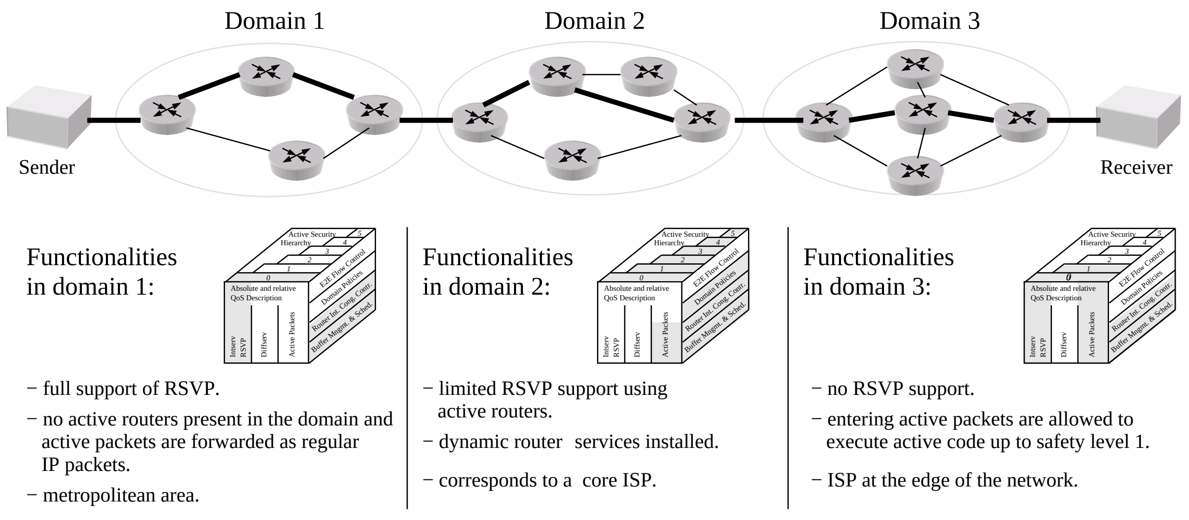

Domain 1
Domain 2
Domain 3
Sender
Receiver
Functionalities in domain 1:
Functionalities in domain 2:
Functionalities in domain 3:
Active Security Hierarchy
0
1
2
3
4
5
Absolute and relative QoS Description
Intserv RSVP
Diffserv
Active Packets
E2E Flow Control
Domain Policies
Router Int. Cong. Contr.
Buffer Mngmt. & Sched.
− full support of RSVP.
− no active routers present in the domain and active packets are forwarded as regular IP packets.
− metropolitean area.
− limited RSVP support using active routers.
− dynamic router services installed.
− corresponds to a core ISP.
− no RSVP support.
− entering active packets are allowed to execute active code up to safety level 1.
− ISP at the edge of the network.

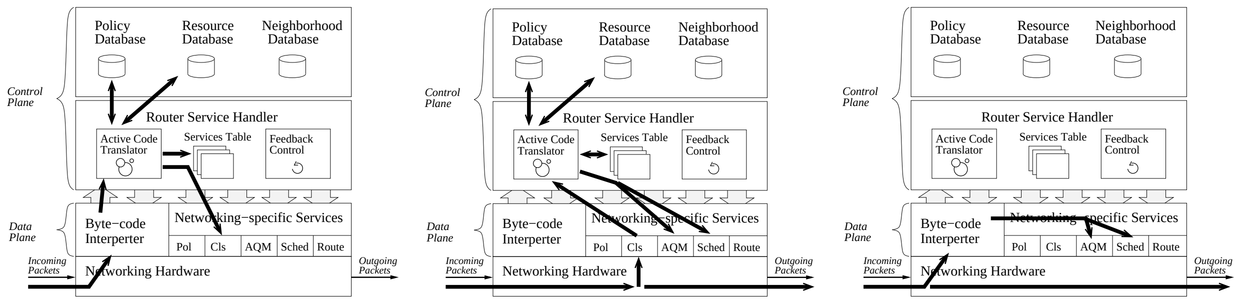
Fig. 6. *End-to-end service in a heterogeneous environment crossing three different domains.*

In domain 2, a core ISP, limited RSVP support is provided using dynamic router services. Active packets that enter this domain and have a safety level higher than 1 are preempted. Preemption can be done by using a preemption flag in the active header, stackable active headers, classification based on the active code, or domain tags derived from an Autonomous System (AS). From the operating expenses and complexity prospect, only the use of a preemption flag is reasonable, although a possible asymmetric behavior has to be accepted. Thus active routers at the ingress of a domain are responsible for setting this flag correctly.

Only the domain administrator is allowed to install dynamic router services on the active routers in his domain. Figure 7.a) shows the sequence executed when adding a new router service. The active packet arriving at the node asks to register a new router service given as payload. As the active packet is not preempted, the byte-code interpreter executes the register request by passing the information to the active-code translator. The active-code translator installs new policies, reserves resources, and, if successful, registers the new service in the service table. At the same time classifiers for the RSVP messages are set up.

In active routers at the edge, RSVP Resv messages are accepted using policies from the policy database. In the core, no further verification of the RSVP Resv messages has to be done, and the content of the message is used to install appropriate filters and flow parameters according to the behavior bounds given in the policy and resource databases (Figure 7.b).

In domain 3, no RSVP support is given, but active packets with security level up to 1 are executed (Figure 7.c). Here, QoS adaptation takes place directly in the data path. Small active-code sections can use the DSCP and local congestion status information to influence the forwarding behavior of the given packet within the limits provided by the ANSB. This ISP provides a slightly larger degree of freedom to applications, but in principle is not willing to guarantee anything more than that. Because of charging of active packets at the edge and the safety properties of the byte-code this network can be exploited with a minium of administration effort. The offered service is comparable to the well known priority service of the post.



(a) Service registration for limited RSVP functionalities in domain 2 using active packets with safety level 5.

(b) RSVP Resv message handled by the ANSB in the core of domain 2.

(c) Active packets with safety level 1 can profit from limited QoS functionalities in domain 3.

Fig. 7. *Sequence of procedures in the ANSB for different packets in domains 1 and 2.*

Although not all domains support RSVP, the end-to-end service is improved, and can in the best case even hide the lack of full RSVP support in all routers on the data path.

6 Summary and Conclusion

This paper shows how the end-to-end quality of service can be improved with active networks. Based on existing QoS capabilities, the active networking approach proposed here provides the necessary tools to dynamically translate between different QoS frameworks and, therewith, enable efficient linkage of QoS parameters to build end-to-end services.

The safety hierarchy introduced consisting of six safety levels provides the necessary safety guarantees and enables dynamic router services for QoS translation in the control plane on the one hand, and, on the other hand, allows the execution of simple active code even in the data path.

In general, information on the network infrastructure and network topologies is not publicly available because ISPs consider this information as sensitive. The use of active networks as proposed in this paper does not expose this information. First, only a part of the available resources can be placed at the disposal of the ANSBs, and second, the safety hierarchy ensures that only information required for a given task is made available, e.g., active code in the data path restricted to safety level 1 cannot use dynamic router services.

References

1. R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview, RFC1633, June 1994.
2. S. Blake, D. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services, RFC2475, December 1998.
3. Y. Bernet, R. Yavatkar, P. Ford, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine. A Framework For Integrated Services Operation Over Diffserv Networks, RFC2998, November 2000.
4. British Columbia Institute of Technology. Bandwidth Broker high level Design, November 1998.
5. R. Guérin and V. Peris. QoS in Packet Networks, Basic Mechanism and Directions. *Computer Networks,* 31(3):169–189, February 1999.
6. B. Schwartz, W. Zhou, A. Jackson, W. Strayer, D. Rockwell, and C. Partridge. Smart Packets for Active Networks. *ACM Computer Commun. Rev.,* January 1998.
7. J. T. Moore, M. Hicks, and S. Nettles. Practical Programmable Packets. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01)*, April 2001.
8. R. Keller, S. Choi, M. Dasen, D. Decasper, G. Fankhauser, and B. Plattner. An Active Router Architcture for Multicast Video Distribution. In *Proceedings of INFOCOM 2000* , March 2000.
9. R. Haas, P. Droz, and B. Stiller. A Hierarchical Mechanism for the Scalable Deployment of Services over Large Programmable and Heterogeneous Networks. In *Proceedings of International Conference on Communications (ICC)*, June 2001.
10. D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing SNMP Management Frameworks, RFC2571, April 1999.
11. Dave Katz, Derek Yeung, and Kireeti Kompella. Traffic engineering extensions to OSPF, internet draft draft-katz-yeung-ospf-traffic-06.txt, October 2001.
12. Klara Nahrstedt and Jonathan M. Smith. The QOS broker. *IEEE Multimedia*, 2(1):53–67, 1995.
13. Jonathan T. Moore. Safe and Efficient Active Packets. Technical Report MS-CIS-99-24, University of Pennsylvania, DSL, October 1999.
14. Andreas Kind, Roman Pletka, and Burkhard Stiller. The Potential of Just-in-Time Compilation in Active Networks based on Network Processors. *IEEE OPENARCH'02*, June 2002.
15. Dan Decasper, Zubin Dittia, Guru Parulkar, and Bernhard Plattner. Router plugins: a software architecture for next-generation routers. *IEEE Trans. on Networking*, 8(1):2–15, February 2000.