Trends in Access Networks and their Implementation in DSLAMs

Christian Sauer, Matthias Gries, Sören Sonntag
Infineon Technologies, Corporate Research, Munich
{christian.sauer|matthias.gries|soeren.sonntag}@infineon.com

Dietmar Tölle, Bo Wu, Rudi Knorr
Fraunhofer ESK, Munich, Germany
{toelle|wu|knorr}@esk.fraunhofer.de

Abstract. We identify deployment trends and primary tasks a future DSL Access Multiplexer (DSLAM) has to offer. We reveal two corner cases: IP-based (high-end) versus Ethernet-based (cost-optimized) access networks. We derive the need for flexible hardware platforms to support fast customization and adaptation to new protocol standards. To accomplish efficiency and ease-of-use we employ programmable multiprocessor platforms and use our tool flow CRACC that takes a modular application description and generates code for various embedded processors.

Keywords: Quality of Service, Benchmarking, Traffic Scenarios, Embedded Systems, Network Processing.

1. Introduction

DSLAMs connect individual customers with the broadband service provider network. They are built modularly out of different line and trunk cards. Line cards aggregate xDSL lines, trunk cards aggregate multiple line cards and provide access to the service provider network. A backplane or dedicated links connect line and trunk cards. Most of the traditional access network equipment will become obsolete in future. Narrowband and ATM switches will be replaced and broadband remote access server (BRAS) functions will be transferred towards the customer ports. We recognize two deployment trends in access networks, particularly for the last mile: Layer 3 is based on IPv4 (IPv6 in the future), layer 2 on Ethernet. The future DSLAM represents the central access point for all narrow- and broadband services and maps all services on IP and Ethernet, respectively. These services include QoS distinction and multicast.

A cost-sensitive solution will use commodity parts that are solely based on Ethernet. Traffic management and QoS are supported but limited to information available in the Ethernet frame, e.g. the addresses and VLAN tag. High-end solutions can afford to parse higher protocol layers and use mechanisms like DiffServ and IPSec. Thus, it is possible to support a wider range of QoS features and use traffic management more adaptively.

Ideally, both cases can be implemented employing different versions of the same scalable platform, thus also reusing the same software development system. In addition, to support fast deployment, protocol adaptation, and design space exploration a modular domain-specific software development framework is required to allow implementing network applications efficiently while abstracting from details of the underlying hardware for portability and ease-of-use. Such a framework should not only allow exploring design trade-offs on a line or trunk card architecture, but also support the quantitative evaluation of functional partitionings between these cards.

In the following, we describe primary tasks of future DSLAMs and sketch the underlying ideas of our software development framework for the successful implementation and deployment of the DSLAM.

2. Primary Tasks of Future DSLAMs

The functionality of a future DSLAM can be described by a set of 13 primary tasks. For the high-end solution, these tasks are described in [3]. We focus on the distinctive primary tasks for the cost-efficient solution in this section. We show at the end, which tasks must be supported by the low- and high-end DSLAM variants.

- *Ethernet tagging/untagging:* Assigns ingress Ethernet packets with appropriate VLAN tag: Single tag for untagged traffic; provider tag for VLAN stacking.
- *Frame header check and update:* Ethernet frame headers are checked and filtering tables are updated.
- *Traffic classification/Forwarding lookup:* Based on preset Ethernet source/destination addresses, VLAN tags, and physical ports a classification is performed to determine traffic class and forwarding information.

Table 1 shows an example, how the described primary tasks can be used to implement a *distributed* DSLAM (high-end [IP] and cost-efficient [ETH] variants). The deployed tasks for line (LC) and trunk cards (TC) are compared. In a distributed implementation most of the processing is done near the ingress point of the traffic. Alternatives are a *centralized* implementation (most processing on trunk card) and a *de-centralized* implementation (most processing on line cards). In the high-end solution, customers are not trusted and an expensive classification and policing step is done on the line cards. Contrary to that, the Ethernet-based cost-optimized solution relies on proper traffic marking by the customer premises equipment. The final decision for one implementation depends on many factors, such as network infrastructure, provider services, and design constraints on costs, performance, and reliability.

Table 1. Deployed functions for distributed DSLAM: Line (LC) and trunk card (TC), high-end (IP) and cost-efficient (ETH).

DSLAM Function	*IP*		*ETH*	
	LC	*TC*	*LC*	*TC*
Ethernet deframing	+	+	N/A	N/A
IP header check	+	+	–	–
IP source address/DSL port verification	+	–	–	–
Ethernet tagging/untagging	–	–	+	(+)
Ethernet header/DSL port verification	–	–	+	–
Traffic classification/Forwarding lookup	+/–	+/+	+/+	–/–
Traffic policing and QoS	+	–	–	–
Multicast duplication	+	+	+	–
Queuing by traffic classes	+	+	+	+
Priority scheduling	+	+	+	+
Set DiffServ codepoint	–	+	–	–
Decrement TTL/HLIM, update CRC	+	+	–	CRC
Ethernet framing	+	+	N/A	N/A

3. Design of a DSLAM

3.1. Functional Implementation

We model the functionality of the future DSLAM in Click [1], a domain-specific framework for describing network applications. Click models are modular, executable, implementation independent, and capture inherent parallelism in packet flows and dependencies among elements. A functionally correct model of the application can be derived quickly. The subsequent performance optimization can focus on individual elements and the partitioning of elements onto processing cores. We have extended the Click library with access network- and Ethernet-specific elements as well as with layered traffic sources.

Figure 1 shows the Click description of an Ethernet based line card as an example. The setup of filter tables, classifiers, and traffic rates together with packet sources and sinks defines a particular environment for the DSLAM.

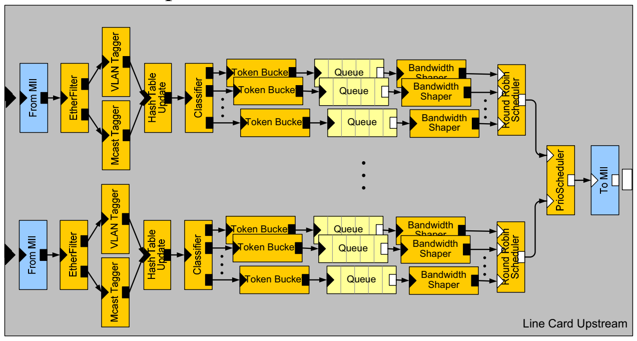

Figure 1. Ethernet DSLAM line card in Click.

The pure Click description of an exemplary distributed DSLAM requires only 266 lines-of-code (w/o comments), representing 932 connected Click elements for four line cards, one trunk card, four ports per line card, three QoS classes per customer, and layered traffic sources and sinks. A full description of an IP-DSLAM can be found in [3].

3.2. DSLAM Architecture Evaluation

We have modeled several functional partitions between line and trunk cards in Click [1] and use our design flow CRACC [2] to generate code for a set of ten embedded processors, including MIPS, ARM, and PowerPC. We use C-compilers and cycle-accurate simulators for the performance evaluation of different partitions.

Since most embedded processors can only be programmed in C, we have developed CRACC, a framework that allows us to generate C from Click's C++ descriptions. The application programmer models the functionality on any host where Click can be simulated. CRACC's Click front-end is used to generate a netlist and the corresponding configurations for CRACC library elements. The source code can be cross-compiled and profiled on the respective embedded platform. Details on CRACC can be found in [2] where we show that a full high-end DSLAM can be implemented with less than 20KB of code memory. As shown in [3], CRACC allows us to explore different functional partitionings and design alternatives in short development time.

4. Concluding Remarks

We have captured trends in access networks, such as QoS per customer, multicast, and the prevalent use of Ethernet/IP end-to-end. We modeled the function of future Ethernet and IP based DSLAMs in an extensible way. Comparing the two implementation variants we find some common features in the data plane. The high-end IP version requires more computational efforts due to richer more adaptive QoS features. The cost-efficient Ethernet implementation relies on pre-computed and configured management information and thus can be implemented with fewer resources. We are currently working on incorporating BRAS functions.

We notice a large productivity gain from using a modular tool flow for applications. A DSLAM scenario can be input within a day, and testing and configuration of filters and lookup tables take another few days. This enables rapid prototyping, whereas in plain assembly a single implementation may take months. Using a library-based and modular ANSI-C tool flow is therefore a reasonable trade-off between implementation efficiency and portability.

References

[1] E. Kohler, R. Morris, B. Chen, et al. The Click modular router. *ACM Trans. on Computer Systems*, 18(3), Aug. 2000.

[2] C. Sauer, M. Gries, and S. Sonntag. Modular domain-specific implementation and exploration framework for embedded software platforms. In *DAC*, June 2005.

[3] C. Sauer, M. Gries, and S. Sonntag. Modular reference implementation of an IP-DSLAM. In *ISCC*, June 2005.