ATM on Linux*

Werner Almesberger
werner.almesberger@lrc.di.epfl.ch

Laboratoire de Réseaux de Communication (LRC)
EPFL, CH-1015 Lausanne, Switzerland

March 11, 1996

Abstract

Since the beginning of 1995, ATM support has been developed for Linux. By now, Linux supports most functionality that is required for state of the art ATM networking. This article introduces to general ATM concepts, presents the current status of development on Linux, and outlines the future direction ATM on Linux will take.

1 Introduction

ATM (Asynchronous Transfer Mode) [1][1] is currently perceived as the most suitable technology for modern high-speed multimedia networks; among other reasons also because its architecture incorporates support for guaranteed Quality of Service (QoS; bandwidth, end-to-end delay, etc.).

In order to create an ATM platform for research and education, the Laboratoire de Réseaux de Communication (LRC) of EPFL is developing ATM support for Linux. In the first project phase, device drivers for two ATM adapters were written, and an extension of the socket API to support native ATM PVCs ("manually" configured connections) was designed and implemented.

Like the telephony network, ATM is connection-oriented and therefore uses signaling to establish ("to dial") connections. In a second project phase, the implementation was extended to support SVCs (connections set up using signaling). Because the protocols involved are rather complex, only a minimum of the functionality is implemented in the kernel and a user-mode signaling demon performs the actual protocol processing.

Since "native" ATM applications are rare and will not be common until ATM is widely deployed, ATM must coexist with "legacy" networks (TCP/IP, IPX, etc.). The IETF and ATM Forum have defined "Classic IP over ATM" [3] and LAN Emulation (LANE) [4, 5], respectively, to carry legacy network traffic over ATM. ATM on Linux also supports IP over ATM for PVCs and SVCs as defined by RFC1577 [6] and others. Work is currently in progress at Tampere University of Technology (Finland) to also support LANE.

2 ATM concepts

ATM is designed for demanding data and multimedia communication, such as audio and video transmission, and high-speed data transfer. The design of ATM has been strongly influenced by the telecommunication community, and therefore ATM is different from data network architectures like today's Internet in many ways. The probably most important differences are the following:

- ATM is connection-oriented
- ATM supports a guaranteed QoS
- ATM clearly distinguishes between end systems and "the network"

*See also `http://lrcwww.epfl.ch/linux-atm/` for the latest status.

1. See [2] for a comprehensive overview of current ATM technology.

All these concepts have their counterparts in the telephony network: you have to establish a connection before you can communicate with the other party, the QoS (i.e. that you get reasonable bidirectional voice transmission) is guaranteed and doesn't depend on the network load, and your telephone is very different from, say, a PBX.

Another difference is that ATM sends data in tiny cells with a fixed size of only 53 bytes instead of in variable-size frames. While this difference is important at the lowest protocol layers, higher layers typically use larger units which are then transformed from/to cells by a so-called "ATM adaption layer" (AAL, [7]).

Figure 1 shows the structure of an ATM network. The network itself consists of interconnected switches. Two types of networks are distinguished: "private" networks are typically company or campus networks, and "public" networks correspond to what is offered by telephone carriers. The standardized interface between end systems ("hosts") and the ATM network is called the "user-network interface" (UNI). The UNI defines several types of physical media (i.e. multi-mode fiber, UTP-5, etc.), many bit rates (ranging from only a few Mbps to 155.52 Mbps and more), line codings, configuration and signaling protocols, etc. The current version of the UNI is 3.1 [8], and version 4.0 is in the final steps of standardization by ATM Forum at the time of writing.

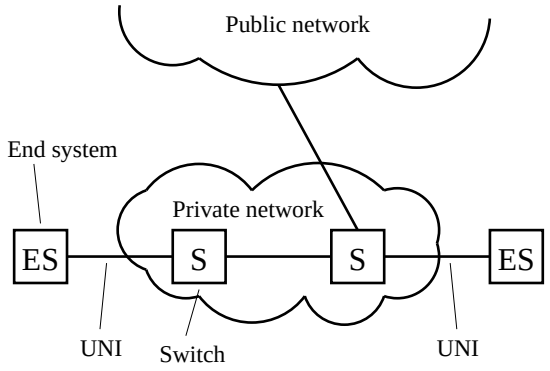

Figure 1: General structure of an ATM network

There are two mechanisms for setting up ATM connections: the simple way is to configure each switch individually (a bit like it was done in the early days of telephony, where operators had to physically connect calls on switchboards). Such connections are called "permanent virtual circuits" (PVCs). A more convenient way of setting up connections it to "dial" them, which is called "signaling" in ATM terminology. "Switched virtual circuits" (SVCs) are set up using signaling. ATM signaling is based on the protocols DSS2 (see Q.2931 [9] for unicast and Q.2971 (not yet published) for multicast), which in turn use the so-called SAAL [10, 11, 12] to transport signaling messages.

Figure 2 illustrates ATM signaling: first, the caller sends a SETUP message towards the destination (1). This message is processed at every single switch. If the destination accepts the call, it returns a CONNECT message (2). Again, this message is seen by all switches. When the CONNECT message reaches the destination, the data connection is established and data can be exchanged between both end systems (3).[2] Note that the switches don't have to interpret what is sent on the data connection.

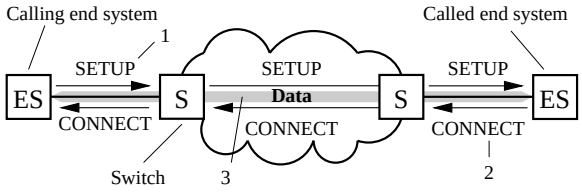

Figure 2: Signaling message flows

3 ATM and the real world

While ATM purists may dream of a world where all computers, TV sets, telephones, etc., are connected to a big ATM cloud consisting of many interconnected ATM networks, the real world is different: connectionless IP networks without QoS concepts play the dominant role, and "native" ATM applications are still a minority.

The first step in running IP over ATM is to have means to carry IP packets on ATM. This is mainly an encapsulation issue, defined in RFC1483 [13]. With this alone, IP can be run over ATM using PVCs.

2. This is slightly simplified. UNI 3.x signaling also allows an acknowledgement for the SETUP message and it requires an acknowledgement for CONNECT.

For SVCs, also a way to resolve IP addresses to ATM addresses is needed. The IETF currently uses an approach called "classical IP over ATM" that is based on an extension of ARP, called ATMARP [6, 14]. ATMARP works like this (see also figure 3): each IP subnet has one ARP server (C). When a client (**A**, **B**) starts, it registers its own IP and ATM addresses at the ARP server (1). Now, if client **A** wants to send data to client **B**, but it only knows **B**'s IP address, it sends an ATMARP request (2) to the server. If the server knows **B**'s addresses, it responds with an ATMARP reply (3), containing **B**'s ATM address. **A** can now establish an SVC to **B** and send data (4).

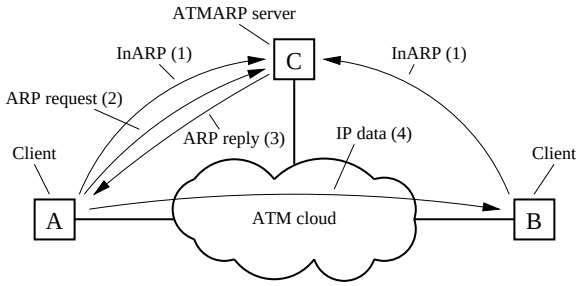

Figure 3: ATMARP message flows

The IETF is working on extending classical IP over ATM to support redundant ARP servers [15], and there is also work in progress on a different protocol called the "next hop resolution protocol" (NHRP, [16]) that allows to establish direct ATM connections even beyond IP subnets. There is also an additional protocol to support IP multicast over ATM [17].

Furthermore, there is work being done on integrating IP mechanisms for negotiating QoS parameters, (e.g. RSVP [18]), with ATM [19, 20].

4 ATM on Linux details

This section describes the development process of ATM on Linux and the current implementation.

Drivers

The first step in bringing ATM to Linux was to find ATM adapters that offered sufficient performance, that were available on the market, and for which programming information was openly available. The search for such adapters turned out to be quite difficult, mainly because at the end of '94, many companies only had products for Sun's SBus, and very few adapters for the PCI bus were available on the market.

Eventually, we chose to use the products from ZeitNet and from Efficient Networks. In spring 1995, a driver for the Efficient Networks EN155p adapter was written, and a driver for the ZeitNet ZN1221 adapter followed soon thereafter. Both adapters are PCI bus cards and run ATM at 155 Mbps over multi-mode fiber.

ATM socket API

In order to send data over even only PVCs, a device driver alone isn't enough, but also an API is needed. Although ATM Forum is defining a semantic API [21], this description is far too general for any concrete implementation. Therefore, based on the BSD socket API, a native ATM API was defined for PVCs and later for SVCs too [22]. This was done in parallel with device driver development.

After some code was written to implement the ATM-specific socket and protocol functions, which interface between the common socket layer and the device drivers [23], early tests were possible. (See figure 4 for the protocol stack.) Since IP over ATM encapsulation is comparably easy to implement, support for classical IP over ATM over PVC was added shortly thereafter.

Figure 4 illustrates the Linux networking protocol stack. The "traditional" IP over Ethernet (or SLIP, PPP, etc.) stack is on the right side, the elements added by the ATM stack are on the left side.

Single-copy

At that time, performance tests revealed that throughput left much to be desired: instead of the theoretical maximum of 135.6 Mbps for user data with raw ATM, only a throughput of approximately 100 Mbps was obtained under ideal conditions. The results for IP over ATM were much worse. The culprit was easily found: because PCs tend to have a

3. This is only an interface to the signaling demon, which performs the actual exchange of Q.2931 signaling messages.

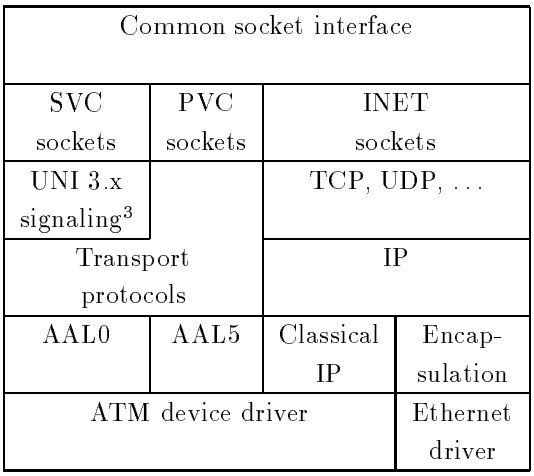

Figure 4: Kernel parts of the Linux networking protocol stack

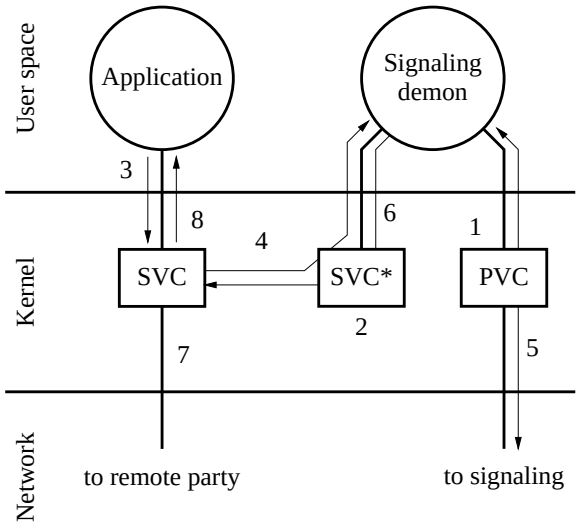

Figure 5: Signaling procedure in the kernel

slow memory interface, the comparably large number of copy operations in the kernel created a bottleneck.

The problem was resolved using a concept called "single-copy", where data is copied directly between user space and the device driver, without additional copying to kernel buffers [24]. With single-copy, transfer rates of up to 130 Mbps are possible on Linux PCs with native ATM when using sufficiently large datagrams.

Signaling

Since PVCs are too inflexible for most purposes, the logical next step was to start to implement signaling. ATM signaling mainly consists of the actual signaling protocol DSS2 and the transport protocol SSCOP [11]. Because those protocols are rather complex but do most of their work only when connections are established or torn down, we decided to implement them in a demon in user mode.

Figure 5 illustrates a typical connection setup: When started, the signaling demon creates a PVC to communicate with the signaling entity in the network (1), and a special SVC socket (2), which is used to exchange signaling messages with the kernel. A very simple protocol is used for the communication between the kernel and the signaling demon.

When an application requests a connection to a remote party (3), the kernel sends a message to the signaling demon (4), which then performs the signaling dialog with the ATM network (5). When the connection is established, the signaling demon indicates this to the kernel (6), which then sets up the local part of the data connection (7) and notifies the application (8). Incoming calls are handled in a similar way.

Later on, a demon was also added for the "interim local management interface" (ILMI, [8]) protocol, which is used mainly for configuration purposes, such as address auto-configuration. This demon was contributed by Scott Shumate of the University of Kansas.

ATMARP

After basic SVC functionality was available, ATMARP had to be implemented to make use of SVCs for IP over ATM too. The approach chosen is similar to signaling: a demon process implements the ATMARP protocol and only a simple table for ARP lookups is kept in the kernel, see figure 6.

When started, the ATMARP demon creates a special socket (1) to communicate with the kernel. When an application wants to send data to an IP destination (2) on the same IP subnet, TCP/IP performs an ARP table lookup (3). If no ATM connection exists for that destination yet, the ker-

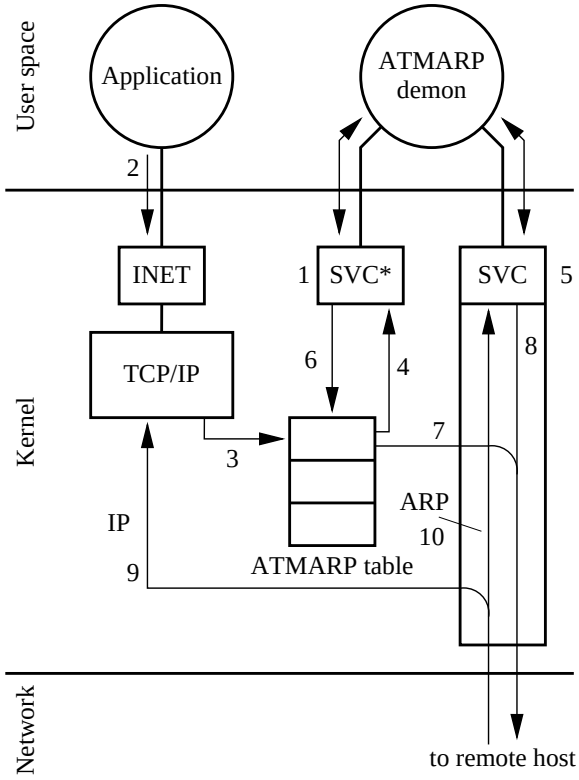

Figure 6: ATMARP procedure in the kernel

nel sends a message to the ATMARP demon requesting resolution of the IP address (4). If the local machine acts as the ATMARP server for the IP subnet, only a lookup in the address resolution table of the ATMARP demon is performed.[4] Otherwise, the ATMARP demon first searches its own table, and if no entry is found, it sends a resolution request to the ATMARP server. If the resolution succeeds, a new SVC is opened for the destination host (5) and it is entered in the kernel ATMARP table (6). Now, IP packets can be sent to the remote host (7).

Note that the ATMARP demon still owns the SVC and that it can send ARP packets to the remote host (8). Also, all incoming packets are examined and they're either sent to the TCP/IP stack (IP, 9) or to the ATMARP demon (ARP, 10).

5 Research uses

ATM on Linux support is developed at LRC as part of our Web over ATM project.[5] One of the first uses of ATM on Linux was for video demos, where images were digitized on SPARCs and then sent over ATM to a PC for displaying.

Because standard IP currently supports neither direct ATM end-to-end connectivity beyond subnet boundaries nor negotiation of QoS aspects, LRC has designed an extension of ATMARP called "application requested IP over ATM" (AREQUIPA) [25]. Arequipa allows applications to request a direct ATM connection for their exclusive use with TCP/IP protocols. A prototype of this will be implemented on Linux.

Other groups have also identified Linux as an interesting platform for advanced research projects, e.g. the U-Net environment [26] is being ported to Linux at Cornell University.

6 The future

The core functionality of ATM on Linux is expected to become stable in spring '96. Arequipa will also be implemented during spring '96. At the time of writing, several people are working on supporting additional ATM adapters on Linux.

The version 4 of the UNI specification will bring many new interesting features, such as an improved multicast concept.[6] It is planned to implement at least some of the signaling improvements of UNI 4.0 until fall '96.

We plan to have complete ATM support on Linux ready for integration into the next stable release of the "mainstream" Linux kernel at the beginning of 1997.

7 Conclusion

A brief introduction to the most important concepts of ATM in today's networking world was given and it was illustrated that ATM on Linux

4. In addition to the entries that are also in the kernel ATMARP table, the ATMARP demon also caches mappings for hosts to which no connection exists.

5. `Seehttp://lrcwww.epfl.ch/WebOverATM/`

6. ATM on Linux doesn't support ATM multicast yet.

supports all the respective mechanisms, and how this is accomplished. For some particularly interesting cases, details about the actual implementation were given.

At the end of this paper, some early research applications and plans for future development were described.

References

[1] Le Boudec, J.-Y. *The Asynchronous Transfer Mode: a tutorial*, Computer Networks and ISDN Systems, Volume 24, Number 4, 1992.

[2] Alles, A. *Internetworking with ATM*, `http://cell-relay.indiana.edu/cell-relay/docs/cisco.html`, Cisco Systems, May 1995.

[3] Cole, R. G.; Shur, D. H.; Villamizar, C. *IP over ATM: A Framework Document* (work in progress), Internet Draft `draft-ietf-ipatm-framework-doc-07.ps`, February 1996.

[4] Truong, H. L.; Ellington, W. W. Jr.; Le Boudec, J.-Y.; Meier, A. X.; Pace, J. W. *LAN Emulation on an ATM Network*, IEEE Communications Magazine, May 1995, pp. 70-85.

[5] The ATM Forum, Technical Committee. *LAN Emulation Over ATM, Version 1.0*, `ftp://ftp.atmforum.com/pub/specs/af-lane-0021.000.ps.Z`, The ATM Forum, January 1995.

[6] Laubach, M. *Classical IP and ARP over ATM*, RFC1577, January 1994.

[7] ITU-T Recommendation I.363. *B-ISDN ATM adaptation layer (AAL) specification*, ITU, 1993.

[8] The ATM Forum. *ATM User-Network Interface (UNI) Specification, Version 3.1*, `ftp://ftp.atmforum.com/pub/UNI/ver3.1`, Prentice Hall, 1994.

[9] ITU-T Recommendation Q.2931. *Broadband Integrated Services Digital Network (B-ISDN) – Digital subscriber signalling system no. 2 (DSS 2) – User-network interface (UNI) – Layer 3 specification for basic call/connection control*, ITU, 1995.

[10] ITU-T Recommendation Q.2100. *B-ISDN signalling ATM adaptation layer (SAAL) overview description*, ITU, July 1994.

[11] ITU-T Recommendation Q.2110. *B-ISDN ATM adaptation layer – service specific connection oriented protocol (SSCOP)*, ITU, July 1994.

[12] ITU-T Recommendation Q.2130. *B-ISDN signalling ATM adaptation layer – service specific coordination function for support of signalling at the user network interface (SSFC at UNI)*, ITU, July 1994.

[13] Heinanen, J. *Multiprotocol Encapsulation over ATM Adaptation Layer 5*, RFC1483, July 1993.

[14] Perez, Maryann; Liaw, Fong-Ching; Mankin, Allison; Hoffman, Eric; Grossman, Dan; Malis, Andrew. *ATM Signaling Support for IP over ATM*, RFC1755, 1995.

[15] Laubach, M.; Halpern, J. *Classical IP and ARP over ATM* (work in progress), Internet Draft `draft-ietf-ipatm-classic2-01.txt`, February 1996.

[16] Katz, D.; Piscitello, D.; Cole, B.; Luciani, J. V. *NBMA Next Hop Resolution Protocol (NHRP)* (work in progress), Internet Draft `draft-ietf-rolc-nhrp-07.txt`, December 1995.

[17] Armitage, G. *Support for Multicast over UNI 3.0/3.1 based ATM Networks* (work in progress), Internet Draft `draft-ietf-ipatm-ipmc-12.txt`, February 1996.

[18] Braden, R.; Zhang, L.; Berson, S.; Herzog, S.; Jamin, S. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification* (work in progress), Internet Draft `draft-ietf-rsvp-spec-10.ps`, February 1996.

[19] Borden, M.; Crawley, E.; Davie, B.; Batsell, S. *Integration of Real-time Services in an IP-ATM Network Architecture*, RFC1821, August 1995.

[20] Birman, A.; Guerin, R.; Kandlur, D. *Support for RSVP-based Service over an ATM Network* (work in progress), Internet Draft `draft-birman-ipatm-rsvpatm-00.txt`, February 1996.

[21] The ATM Forum, SAA API Ad-hoc Work Group. *Native ATM Services: Semantic Description Version1.0*, ATM Forum contribution 95-0008, January 1996.

[22] Almesberger, W. *Linux ATM API*, `ftp://lrcftp.epfl.ch/pub/linux/atm/api/`, EPFL, November 1995.

[23] Almesberger, W. *Linux ATM device driver interface*, `ftp://lrcftp.epfl.ch/pub/linux/atm/docs/`, EPFL, January 1996.

[24] Almesberger, W. *High-speed ATM networking on low-end computer systems*, Technical Report No 95/147, `ftp://lrcftp.epfl.ch/pub/linux/atm/papers/atm_on_lowend.ps.gz`, EPFL, August 1995.

[25] Almesberger, W.; Gauthier, E.; Le Boudec, J.-Y.; Oechslin, P. *Guaranteeing Quality of Service for the Web using Application REQuested IP over ATM*, Technical Report No 95/158, `http://lrcwww.epfl.ch/WebOverATM/about_woa/bb96.ps`, EPFL, November 1995.

[26] Basu, A.; Buch, V.; Vogels, W.; von Eicken, T. *U-Net: A User-Level Network Interface for Parallel and Distributed Computing* (CS-TR to appear), `http://www.cs.cornell.edu/Info/Projects/ATM/unet-tr.ps`, Cornell University, April 1995.