Prototyping Experiences with Classical IP and ARP over Signaled ATM Connections

(Prototyping Experiences with IP over ATM)

Christoph L. Schuba, Eugene H. Spafford

COAST Laboratory, Department of Computer Sciences, Purdue University, 1398 Computer Science Building, West Lafayette, IN 47907–1398

Berry Kercheval

Computer Science Laboratory, Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304–1314

This paper discusses a prototyping effort in the classical IP (*internet protocol*) and ARP (*address resolution protocol*) over ATM (*asynchronous transfer mode*) model. It introduces the important features of the involved protocols, motivates the interaction between IP and ATM, and examines several possible scenarios for it. The design and implementation of a driver prototype for an ATM network adapter are described. The prototype utilizes and dynamically manages signaled ATM connections in a way that is transparent to user processes.

The paper shares our experiences from the design and implementation of this prototype, such as the lessons learned how to deal with problems we ran into that were not anticipated. The lessons of this paper are applicable to a wide range of software prototyping efforts, not only the system described.

⋆ To appear in the Journal of Systems and Software, III/1998.

1 Introduction

The *internet protocol* (IP) is the standard protocol on the Internet that provides the basis for a connectionless, best-effort packet delivery service. IP uses an *address resolution protocol* (ARP) which is based on broadcast to dynamically bind high level protocol addresses to low level physical hardware addresses.

The *asynchronous transfer mode* (ATM) is rapidly becoming acknowledged as the base technology for the next generation of global communications. Both the technology and the standardization process receive extensive acceptance throughout the industry. Some of ATM's main characteristics are point-to-point or point-to-multipoint connection-oriented communications with small cells as data units and a good aggregate behavior. This particular form of cell networking provides the capability for high speed transmission in local area and wide area networks. Another strength of ATM are end-to-end *quality of service* (QoS) guarantees to support applications with special QoS demands.

The combination of the well established place of IP in the data communication world and the telephone companies' need for ATM as their new base technology establish the need for the integration of IP and ATM. In spite of some problems with the technology it promises a considerable service improvement for data communications.

This paper explains one architectural proposal for the interaction between IP and ATM and describes a prototype implementation. The physical topology of ATM networks and the logical structure imposed by the IP model are not easily mapped onto each other. To integrate the two different communication paradigms one can create *logical IP subnetworks* (LIS) in ATM that operate and communicate independently of other LISs on the same ATM network.

A modified form of ARP is used in an LIS to resolve IP to ATM addresses. Before data can be transmitted between two ATM endstations, a connection is established under the usage of a connection management module and data is packaged and fragmented into small size ATM cells by an adaptation layer. If other protocols besides IP utilize the ATM network an additional encapsulation of data becomes necessary to demultiplex received and reassembled ATM cells if the protocols are all multiplexed over a single virtual channel.

2 Integration of Classical IP and ARP with ATM

We outline the main features of the two technologies, establish their importance as data communication paradigms, motivate their integration, and sketch several scenarios of their interaction. We concentrate on one specific scenario: the utilization of ATM as a logical IP subnetwork. Building blocks of this approach are the ATM address resolution protocol as a unicast server based emulation of the classical broadcast address resolution protocol, and connection management functionality as provided by the Q.2931 protocol. The manipulation of transmitted network protocol data units between their submission to the network layer and their transmission by ATM are detailed in an appendix. These explanations are structured analogous to the lower layers in the IEEE local area network model.

2.1 Classical IP and ARP

The *internet protocol* (IP) [Postel, 1981a] defines a best-effort, connectionless delivery mechanism. It provides for the transmission of blocks of data (datagrams) from sources to destinations over an interconnected system of networks. IP imposes few requirements on utilized network technologies. Therefore, IP can only guarantee an earnest attempt to deliver packets. A *local area network* (LAN) protocol defines the physical and *data link layer* (DLL) in the ISO-OSI (*International Standards Organization - Open Systems Interconnection*) reference model.

Concentrating on the IEEE approach of modeling networks (e.g., [IEEE 802.2]), the DLL is divided into *medium access control* (MAC) and *logical link control* (LLC) sublayers. The LLC sublayer provides a common interface for multiple network layer protocols to interoperate with different MAC sublayers and share the use of a data link. The MAC sublayer defines the mechanisms that are used to access, share, and manage a communication medium. That includes channel management, collision detection and resolution, priority handling, error detection and framing. Typical examples of LAN technologies (MAC sublayer and physical layer) are defined by the family of IEEE 802.x standards (e.g., CSMA/CD [IEEE 802.3], or DQDB [IEEE 802.6]). Figure 1 illustrates the correspondence between MAC & LLC and the DLL.

<table>
<tr><td>3</td><td>Network Layer</td><td></td><td></td></tr>
<tr><td rowspan="2">2</td><td rowspan="2">Data Link Layer</td><td>Logical Link Control (LLC)</td><td>IEEE 802.2</td></tr>
<tr><td>Medium Access Control (MAC)</td><td rowspan="2">IEEE 808.3-9</td></tr>
<tr><td>1</td><td>Physical Layer</td><td>Physical Layer (PHy)</td></tr>
<tr><td colspan="2">OSI Layering</td><td colspan="2">IEEE Layering</td></tr>
</table>

Fig. 1. Correspondence between ISO and IEEE layering, in particular the correspondence between data link layer and medium access control and logical link control

In this model two machines on a physical network can only communicate if they know each other's MAC addresses. Determining a machine's MAC address given its network layer address is known as the address resolution problem.

2.2 ATM

Asynchronous transfer mode (ATM) was developed for use in *broadband integrated services digital networks* (B–ISDN) to carry data, voice, images, and video traffic in an integrated manner. ATM is not limited to B–ISDN and contains physical layer and network layer functionality. Its architecture is based on switching small fixed-length packets called cells. Some aspects of ATM are currently defined by interim standards developed by a user and vendor group known as the ATM Forum [Forum, 1996]. ATM provides for point-to-point or point-to-multipoint, connection-oriented transmission of small data units. ATM gives quality of service guarantees (i.e., it handles resource reservation, offers bandwidth and latency guarantees), and exhibits a good aggregate behavior.

Before data can be transferred between machines, a connection must be established. These connections are called *virtual channels* (VC) and are identified hop-by-hop using a *virtual path* and *virtual channel identifier* pair (VPI/VCI). Data are transferred in the form of cells which are small fixed-size data packets with a 48 byte payload and a 5 byte header. Each switch contains mappings of input to output VC identifiers. These can be (semi-) permanently installed, (called *permanent virtual circuits*, or PVCs) or established at connection setup time by a signaling protocol (called *switched virtual circuits*, or SVCs). The switching of cells involves the appropriate change of VC information in the cell header and a forwarding of the cell over the associated physical link. A switch controller is associated with one or more ATM switches. It performs management functions such as enabling or disabling ports, polling for status information, or updating mapping information.

2.3 Motivation for the Integration of Paradigms

IP and ARP (*Address Resolution Protocol*) on the one hand and ATM on the other represent very different paradigms for data communication. What is the motivation to integrate these two technologies?

IP is the main building block of the Internet, the global network of interconnected networks. In the classical TCP/IP protocol hierarchy (see [Comer, 1991a]) IP is the central protocol. It serves a hierarchy of protocols (e.g., ICMP [Postel, 1981b], EGP [Mills, 1984], UDP [Postel, 1980], or TCP [Postel, 1981c]), and integrates a wide variety of network technologies as its underlying data link and physical layers. Its main concepts have ripened to maturity and proven adequate for robust data delivery in interconnected heterogeneous environments. IP has established its place in today's data communication world. That is not expected to change in the foreseeable future.

ATM is rapidly becoming acknowledged as the base technology for the next generation of global communications. Telecommunication industries are planning on implementing ATM because it meets internal telephony needs. The integration of voice and data traffic into one medium saves resources. It is easier to multiplex separately addressed packets of voice and data than to multiplex mixes of individual bits. Moreover, new services can be offered to users, and the technology can be used to satisfy the telephone companies' own need for increased data communications because of increased computerization. ATM deployment in the local area promises performance advances in distributed computing with high bandwidth and low latency data communication requirements, such as required by some multimedia or interactive applications. Furthermore, ATM as a LAN technology promises easy connectivity of existing hubs, bridges, and routers to future wide-area ATM networks.

ATM is not without its problems or detractors, however. For example, it is not clear that classical telecommunication traffic and data communication will integrate well. Classical telecommunication can be described well by Poisson models, whereas the burstiness of self-similar data traffic does not change over time (see [Leland et al., 1993]).

There are difficulties with scaling ATM and maintaining reliability. Virtual circuits establish a *hard state* in the network for a call. The state of a connection is distributed over intermediate switching equipment. Once a VC is established, it is maintained until a message is received by one of the ends of the call requesting a change in the state of the connection. The product law of reliability ([Trived, 1982, Ch.1.10]), applicable to series systems of independent components, such as the the links in a virtual circuit, demonstrates how quickly system reliability degrades with an increase in complexity. An ATM connection is a series system, because the hops of a connection are so interrelated that no data can be delivered if one link fails. Define for $i = 1, 2, ..., n$: A_i :=“Link i is functioning properly.” and R_i :=“Reliability of link i, i.e., probability that A_i functions properly”. The probability R_s that the system is functioning properly is then:

$$R_s = P(A_1 \cap A_2 \cap \cdots \cap A_n) = \prod_{i=1}^{n} R_i$$

This demonstrates how quickly the connection reliability degrades with an increase in the number of links.

On the other hand parallel redundant systems have a reliability that approaches 100%:

$$R_p = 1 - \prod_{i=1}^{n}(1 - R_i)$$

The topology of IP internetworks and the capability of dynamic rerouting in a highly redundant system such as the IP network layer results in a series-parallel system that approaches the reliability of a parallel redundant system (see [Trived, 1982, Ch.1.10] for an analysis of the reliability in series-parallel redundant systems). ATM does not yet support any dynamic routing functionality or network admission control.

For a more detailed discussion of benefits and drawbacks of ATM see [Partridge, 1993, sections 4.2 and 4.10] and [Bertsekas and Gallager, 1992, section 2.10]

The combination of the well established place of IP in the data communication world and the telephone companies' need for ATM as their new base technology establish the need for the integration of IP and ATM. In spite of some problems with the technology it promises a considerable service improvement for data communications.

2.4 Scenarios of Interaction

The *IP over ATM* Working Group of the *Internet Engineering Task Force* (IETF) is chartered to develop standards for routing and forwarding IP packets over ATM subnets. In [Cole et al., 1996] the working group identifies and discusses several IP over ATM models: the classical IP over ATM model, the *routing over large clouds* (ROLC) model, Ohta's "conventional" model, peer models, the *private network network interface* (P–NNI) and integrated models, and transitional models.

The remainder of this paper focuses on the classical IP over ATM model. We give a brief description of the other models here.

The ROLC model supports SVCs as well a PVCs. It is currently work in progress in the ROLC working group in the IETF. Endstations do not necessarily share a common IP level network prefix, and multiple SNAP address formats and negotiation of parameters (such as MTU (*maximum transfer unit*) size or method of encapsulation) are supported. The associated *next hop routing protocol* (NHRP) performs address resolution to accomplish direct connections across IP subnet boundaries. The fact that ROLC based networks are not part of a single administrative group creates several interesting issues, such as security considerations and the need for a billing scheme.

Ohta's "conventional" model has the same network layer architecture as the standard IP model, including its subnet architecture. However, IP level routers are assumed to be able to forward data cell by cell, yielding a possible latency advantage ([Cole et al., 1996, §8.3]).

Peer models are motivated by the rationale that routing complexities of future ATM networks will be similar to routing over complex internets. The possibility to embed IP level routing within the ATM network fabric leads to peer models where IP routers are placed on a peer basis with corresponding entities in an ATM network ([Cole et al., 1996, §8.4]).

The P–NNI and integrated models are based on the idea that a single routing protocol be used for both IP and ATM. These protocols take into account the IP and ATM network topologies, link state and node characterizations, and calculate optimal routes for IP packets over heterogeneous topologies ([Cole et al., 1996, §8.5]).

Transitional models are a compromise between the classical IP models and the peer and integrated models. They realize the need for heterogeneous environments with complex routing topologies. The success of new technologies often depends on their capability to be gradually introduced and coexists with deployed technology. Transition models address this issues.

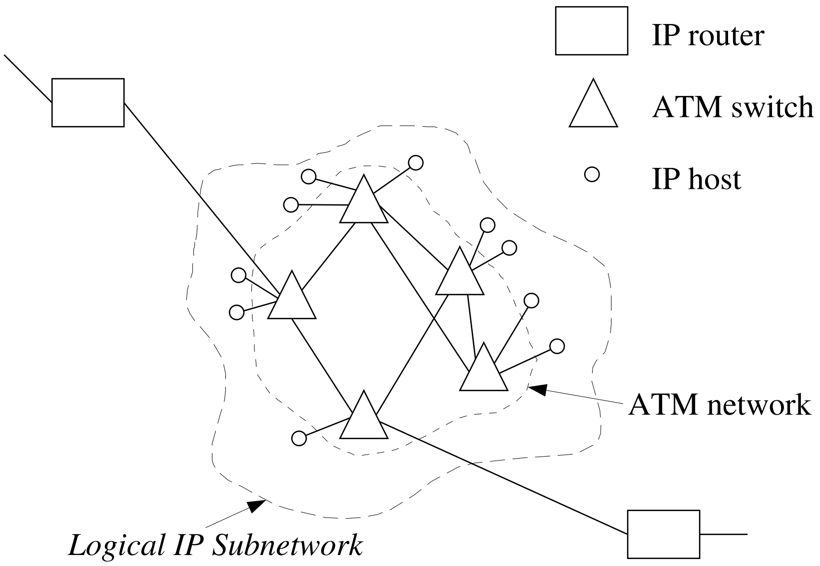

Fig. 2. A logical IP subnetwork (LIS or LATM) appears to the IP layer as just another LAN technology

The classical IP over ATM model takes advantage of the fact that IP can make use of any networking technology that provides connectivity as a delivery subsystem, including ATM. The goal of this approach is to transmit IP datagrams over an ATM LAN (see Figure 2). A *logical IP subnetwork* (LIS or LATM) is a single ATM network, that is a direct replacement for the local area network segment connecting IP routers and endstations in the classical IP and ARP paradigm. All endstations are directly connected and they share a common IP level network prefix.

One LIS can grow to replace several LAN segments. Several LISs can operate on the same ATM network. At least one IP router must be attached to each LIS to integrate it with the rest of the internetwork. It is configured as a member of one or several LISs and handles inter domain routing of datagrams transparently. IP hosts on the same ATM network that are configured to belong to different LISs must not establish a connection to each other directly through ATM, but must communicate with each other via an intermediate IP router.

The previously mentioned necessity for the network layer to resolve its addresses to MAC addresses (see section 3) is not the only problem that has to be solved to establish an LIS as a new subnetwork technology. In appendix A we will explain the conversion between network layer protocol data units (e.g., IP datagram) and ATM cells and the need for adaptation and encapsulation layers and a convergence sublayer. Section 4.1 will outline our approach to an integrated address resolution and call establishment in the presence of switched virtual circuits.

3 The Protocols

Address resolution in an LIS, and ATM connection management are nontrivial protocols. In this section we are concerned with their description.

3.1 ARP and InARP

The address resolution problem, e.g., determining a machine's MAC address given its network layer address, has been solved in many different ways. One well known solution in the case of Ethernet as the LAN technology is ARP

([Plummer, 1982]). A machine S determines the MAC address of another machine D by broadcasting an ARP request containing D's network protocol address. The machine that matches the network protocol address of an ARP request responds with an ARP reply containing D's MAC address.

Inverse ARP[1] (InARP) is defined in [Bradley, 1992] and addresses a different problem. Given an established connection between two machines, they do not have to know one another's higher level protocol addresses if the call establishment was based on lower layer addresses. InARP is a protocol that enables either endpoint of a connection to query the other one for its protocol addresses.

3.1.1 Why Classical ARP is not sufficient

Classical ARP relies on an efficient broadcast capability of the physical layer[2]. Even if ATM multicast were already widely implemented, it is not advisable to simulate broadcast by ATM's multicast connections, because of its poor scaling. Point-to-multipoint connections offer bidirectional communication only between the originator and all destinations, but not among the destinations. To simulate broadcast, each endstation would have to establish a point-to-multipoint connection to all other endstations.

A different approach, described in [Laubach, 1994], is to use a dedicated service on the network that maintains a cache of address mappings. The mappings are established through registration by each host at its boot time and are periodically updated. The service replies to address resolution requests with

[1] Note that we are talking about InARP, not *reverse ARP.* (RARP) [Finlayson et al., 1984].

[2] We are not considering a static address binding for ATM networks, because ATM physical addresses are longer than IP addresses and cannot be freely chosen.

the desired mapping, or a negative acknowledgment, if the mapping is not available.

ATM physical addresses are longer than IP addresses. Therefore IP cannot use static address binding for ATM networks.

3.2 ATMARP and InATMARP

The *ATM address resolution protocol* (ATMARP) is basically the same protocol as ARP extended with support in a unicast server environment. ATMARP answers the question: "Given a network protocol address, what is the associated ATM address?".

Inverse ATMARP (InATMARP) is the same protocol as InARP applied to ATM networks. Given a virtual circuit identifier between two endpoints, InATMARP answers the question: "What are the other endpoint's network layer and ATM address?" All hosts participating in an LIS must support these extended protocols as defined in [Laubach, 1994]. Each LIS must have one ATMARP server.

The ATMARP service contains the following elements which we will outline in successive sections: address mapping registration, queries and positive or negative replies, and mapping table maintenance.

3.2.1 Registration

ATMARP clients must start the ATMARP registration process with the server before they can participate as a member of an LIS. Each client must initiate a SVC connection to the server. Upon ATM call acceptance from any other ATM endpoint, the server generates and transmits an `InATMARP_REQUEST` to the client asking it for its network layer to ATM address binding. Clients must reply to each `InATMARP_REQUEST` with an `InATMARP_REPLY`. The server will update his ATMARP table with the received mapping and a timestamp unless a mapping for the same network layer address and an established VC already exist.

3.2.2 Query and Reply

Once hosts in an LIS are registered they can request address resolutions for other hosts in the same LIS via `ATMARP_REQUESTs`. Clients generate and transmit `ATMARP_REQUEST` packets to the server. An `ATMARP_REPLY` packet from the server then contains the mapping which is used to build/refresh the clients cache. An `ATMARP_NAK`[3] packet from the server tells the client that the requested mapping could not be resolved. It is an extension to classical ARP adding to the robustness of the protocol by giving clients the ability to distinguish a cache miss from a fatal server failure.

3.2.3 Cache Maintenance

The ATMARP server maintains an ATMARP table that ideally contains the network layer to ATM address mapping for all hosts participating in its LIS. It is initially built by the `InATMARP_REPLY` packets received from hosts at registration time. Clients maintain a similar table that contains mappings that they previously requested from the server. The entries in the tables are aged periodically. [Laubach, 1994] requires that client table entries remain valid for at most 15 minutes and server table entries for at least 20 minutes. Before invalidating an entry that still has an open VC associated, a server must attempt to

[3] *nak* is the abbreviation of *negative acknowledgment*.

revalidate the mapping by generating and transmitting an `InATMARP_REQUEST`. Entries without an associated open VC are deleted. ATMARP table entries exist until they are aged or invalidated. Teardown of an SVC does not remove ATMARP table entries.

There are other occasions when the server can update his table. When the server receives an `ATMARP_REQUEST` over a VC, whose associated network layer and ATM address match the table entry, the server can update the timeout on that entry. To add robustness, the server examines the source information of received `ATMARP_REQUEST`s. If there is no table entry present for the source network address, the server adds the source information mapping to its table associated with the VC.

3.2.4 Unresolved Issues

Several issues remain unresolved by [Laubach, 1994]. If only one server per LIS assumes the task of address resolution, the service depends on the overall reliability of that server machine. A protocol that embodies a replicated ATMARP service is desirable. Currently the ATMARP service is reached via a well known address. A mechanism should exist for determining the ATMARP service access point dynamically.

3.3 Connection Management

Communications in ATM are based on a connection-oriented model. Before user data can be transmitted, a virtual circuit has to be established. VCs come in two forms: permanent VCs, and signaled VCs. Permanent VCs are statically configured and installed into the switching tables of the ATM switches. Signaled point-to-point and point-to-multipoint connections over an ATM network are dynamically established, maintained, and cleared on a need basis utilizing a signaling protocol, such as the Q.2931 protocol (see [Forum, 1994]), a broadband signaling standard.

The initiator of a connection specifies desired characteristics for the connection and relies on Q.2931 to establish or to report the failure of establishment of a call. Q.2931 neither supports any routing functionality nor provides for call acceptance or forwarding policies.

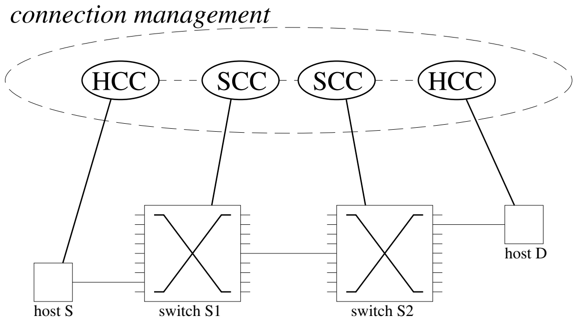

Fig. 3. ATM connection management

Each entity participating in connection management must have an associated process that handles Q.2931 protocol messages. We distinguish between *host call control* (HCC) and *switch call control* (SCC) processes. Figure 3 depicts the interrelation among the switching processes and their relationship to either ATM switches or end systems. All HCC and SCC processes have some state about local network configuration, such as port identification, routing information, and address information for neighboring HCC and SCC processes. This state can be acquired through static configuration, or dynamic discovery and update protocols. HCC processes can initiate the establishment and release of calls. A HCC process communicates with the application or the internetworking sublayer and the Q.2931 module to coordinate their interaction regarding offering, acceptance, and release of each call. A SCC process is responsible for interfacing peer Q.2931 instances on the ATM switches. It accepts incoming call requests from other SCC or HCC processes and determines the outgoing port which should be used when routing the call. Routing information is accessed through a route manager on the ATM switch.

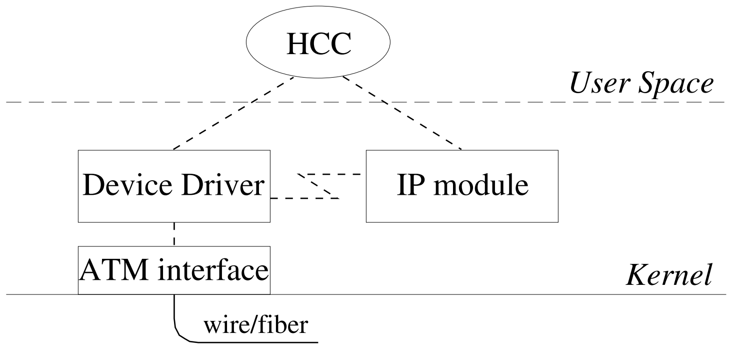

Fig. 4. Integration of connection management into a host

Figure 4 sketches the integration of the connection management into a host. An HCC module is associated with each ATM interface. It interacts with the ATM interface to learn about requirements for the establishment or release of connections. It communicates with its peer SCC and HCC processes on remote switches and hosts over well known signaling PVCs.

4 Experiences and Lessons

We implemented a prototype of this system under SunOS 4.1.3 for ForeRunner SBA–100 and ParcNic ATM host adapters. The connection management code used was an implementation of Q.2931 by Bellcore, called Q.Port (see [Bellcore, 1995]). During the development we learned the following lessons.

4.1 Model

> It is advantageous to develop a model one can follow closely during the implementation to convince oneself of its completeness and correctness, and facilitate its stepwise testing and extension.

In the previous sections we have established the complexity of classical IP and ARP over ATM in an SVC–based subnet as a whole. To deal with the complexity of the functional requirements it is important to devise a model that guides design and implementation of the system. We will describe several additional advantages that a good model can contribute to the design and implementation process.

Even from the limited perspective of one participating host a variety of functions need to be performed. In our example, the system has to integrate connection management, address resolution, protocol and adaptation layer encapsulation, as well as segmentation and reassembly. The system has needs to handle exceptions such as expired timers for incomplete reassembly, or temporary buffering of outgoing packets for which establishment of an SVC is in progress.

The functional modules in ATM as a LAN technology contribute actions as well as pieces of information. It is possible to describe actions, e.g., call establishment, or address resolution, that have taken place with the amount and relationship of address information that has been gathered. That means the interaction between signaling protocol and address resolution protocol in an LIS can be reduced to a conceptually much simpler problem, which is described by a finite state automaton. The states in our model are the possible combinations of network layer address, medium access control sublayer address, switched virtual circuit identifier, and signaling code specific call handle in regard to one destination network layer address. The set of possible states is identical for the ATMARP server and its clients (i.e., all participants of an LIS). The transitions are fired by events based on user data traffic, connection protocol messages, address resolution protocol messages, or timer expiration.

Figure 5 serves as an example for the described automaton. It shows a subset of the automaton that is required to describe the complete system. This subset is sufficient for understanding of our approach. The complete automaton can be generated by applying all possible events to an initially empty state (see state (0) in Figure 5 and all resulting states, until a complete transition table is created. New states and actions associated with the state transitions are determined by careful study of the involved protocol specifications.

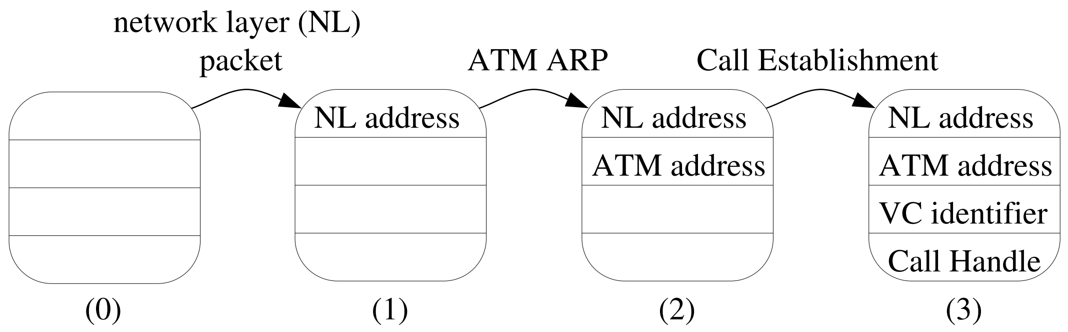

Fig. 5. Example for a complete resolution process. This graph is a subset of the real automaton

Figure 5 depicts a scenario where a network layer packet is passed to our module and neither a known ATM address mapping, nor an established switched virtual circuit to the destination address is present (State (0)). The first transition results in buffering the packet, sending an `ATMARP_REQUEST` to the ATMARP server, and recording that this request was sent (State (1)). If the `ATMARP_REPLY` contains the desired mapping, it moves the automaton into a state where the mapping network layer address to ATM address is complete

(State (2)). A fresh timer value will be associated with this mapping (not shown in Figure 5) and the recorded request will be cleared. The next action at this point is to send an `attempt` message to the HCC and record this attempt. Once an `originated--call--connected` message from the HCC is received, we move into the fully resolved state, store the associated VC identifier and call handle, clear the recorded attempt, and send all network layer packets that were enqueued up to that point (State (3)). All successive packets to the same address will be sent over the associated SVC directly.

The automaton exists concurrently in many different instances in a host: once for each destination network layer address. Ideally this automaton has to be augmented to note which encapsulation type is associated with each connection. As explained in appendix A.1, our approach assumes that LLC/SNAP encapsulation is present. State (0) as in Figure 5 is the initial state. The set of final states is empty because this protocol does not have a natural termination. The initial state (0) never needs to be instantiated until a transition to state (1) is fired, because no data needs to be stored for state (0).

We chose this model because it has several nice features. Its simple structure is easy to understand. It instruments a structured implementation and assists the designer in convincing himself of its completeness and correctness. Furthermore, the model facilitates the stepwise testing and extension of the implementation. That is particularly important if the prototyping effort is undertaken by a project team, rather than a single person. For a functionally restricted prototype, a partial implementation of the automaton is sufficient. A stepwise extension is generally possible without the modification of the already implemented parts.

4.2 Modloadable Driver Software and Modularization of Code

For code development in driver modules it may be advantageous using the capability of dynamically loadable kernel modules at runtime, if available on the development platform.

Several operating systems, e.g., SunOS, have the capability to link object code into the kernel at runtime without halting or rebooting the machine. This capability can save time. Time for compilation is saved if only the driver code and not the whole kernel need to be recompiled for a test run. Time consuming system reboots become less frequent. Usually, little work is necessary to write wrappers to make modules (un)loadable. Commands exist for loading, unloading, and displaying the status of loaded modules.

In our design all system components, except for the implementation of the signaling code were contained in the loadable module. That means our prototype contributes considerably to the runtime size of the kernel. A production version of this project would have to move as many components of the system outside the kernel and keep only core functionality in the kernel that is performance critical, such as runtime IP address to VC identifier lookup. This approach offers the promise of reusing user level components of the implementation for ports to different hardware platforms. It also makes user level development environments available during the implementation, which are often “better” than the tools available for kernel development.

4.3 Separation of Development and Test Platform

It is advantageous to make a conscious decision at what time to use the development platform as the test platform, and at what time to separate the two.

In our example we could use the ATM interface as a secondary network interface on a development machine, or on a different machine dedicated to testing.

In an early stage of the development of low level software, such as device drivers, frequent crashes of core system software components need to be expected. Crashes can cost considerable time to restart a typical development machine to its pre crash setup. Furthermore, in today's computing environments, it is not uncommon that remote users depend on network services or file systems offered by the development machine, in particular if the development is a group effort.

At an advanced stage of the development process unifying development and test machine can be advantageous. It provides the developer with day to day usage experience under realistic working conditions, and possibly even provides additional motivation to do a really thorough job, because his work environment directly depends on the quality of his product.

4.4 Clear Text Kernel Debugging Interface

Adding a clear text debugging interface to low level modules is a very powerful mechanism.

One of the main disadvantages in low level system code development is the limited availability of good development tools. A simple, yet very powerful mechanism, is to add a device file interface (e.g., `/dev/atm_debug`) to the driver that supports file system primitives, such as *open()*, *close()*, *read()*, *write()*, *select()*, and *ioctl()*. We implemented a simple parser in the driver that could act on "input" from a user space application (called "tuning application"). This allowed us to download all kinds of testing parameters (e.g., connection states or ATMARP database state) into the driver and to upload debugging output and data structure contents for closer examination. Our driver performed its own memory management, so it was possible to reset all available driver data structures to initial or previously saved states. Test runs could be automated and misbehavior reconstructed for closer examination. An advantage of this approach over the usage of a kernel debugger is that the interaction with the driver's internal debugging code can easily be automated and used by shell or Perl scripts.

UNIX System V offers the process file system mechanism to read and write the virtual address space of any process. The shortcomings of this mechanisms for our purpose is that kernel modules usually run on behalf of some process or an interrupt handler. It is difficult to keep track which process will be the right one to examine. Furthermore, an interface to the module allows to directly invoke actions that are implemented in the module.

4.5 Graphical User Interface

> Adding a graphical user interface (GUI) to drive the test environment can improve the ease of testing, and increase productivity and quality of the developed software.

The use of a graphical user interface (GUI) as a frontend to the UNIX style command line interface of the configuration application not only increased our productivity, but also aided in the deployment of the software. We extended the debugging interface to be regularly used for driver configuration. The initial UNIX style command line interface of the configuration application became too tedious and time consuming. We wrapped a graphical user interface around the configuration application. Tcl/Tk was used for building the GUI. We were impressed by the little amount of time and code required to build a fairly sophisticated user interface with logging and help text facilities.

It increased productivity of the developers because it offered an easy to use interface for repetitive debugging and configuration commands. It is considerably easier to select a few options with a mouse and click on a "commit" button than to type cryptic command line options. We believe the GUI also aided in the adoption of the software by the test group, because of the ease of configuration, availability of "help" information and documentation for each option. Additionally, such a command line frontend makes it possible to collect action logging for investigation of misbehaving systems.

4.6 Future Model Improvements

In section 4.1 we already described a simple model that we used during the design and implementation phase of the project. Again, the model instrumented a cleanly structured implementation and assisted us in convincing ourselves of its completeness and correctness. Furthermore, the model facilitated the stepwise testing and extension of the implementation.

Although the model proved to be very useful, we have since discovered that it should be applied differently. We had modeled the relationship between IP addresses, ATM addresses, and VCs as one mapping entity, which caused quite some redundancy. The relationship should rather be modeled such that states are represented by data entries in a relational fashion. The relations must describe the functional dependencies among the different address and identifier entities that are mapped onto each other. The basic idea here is that the mappings can be described without loss of information in 5NF[4], not in

[4] NF stands for *normal form*. See [Date, 1991, chapter 21] for a description of

1NF such as in our model. The advantages are a reduction of the degree of redundancy in the state information, a smaller automaton and therefore easier model, and the possibility of a simplified enforcement of integrity constraints. This proposal for future improvements was not implemented.

4.7 Performance

We tested the performance of our system. The test configuration consisted of a SparcStation 10 (SS10) with the experimental ParcNic card, one or two PARC-built ATM (BADLAN) switches and either a SparcStation-20 (SS20) with a prototype Sun ATM card (SAHI) or a SparcCenter-1000 with a Fore SBA-200 card (see Figure 6). The prototype driver was not extensively tuned for best performance; effort was concentrated more on correct behavior and robustness than on raw speed, as it was desired to use this code daily.

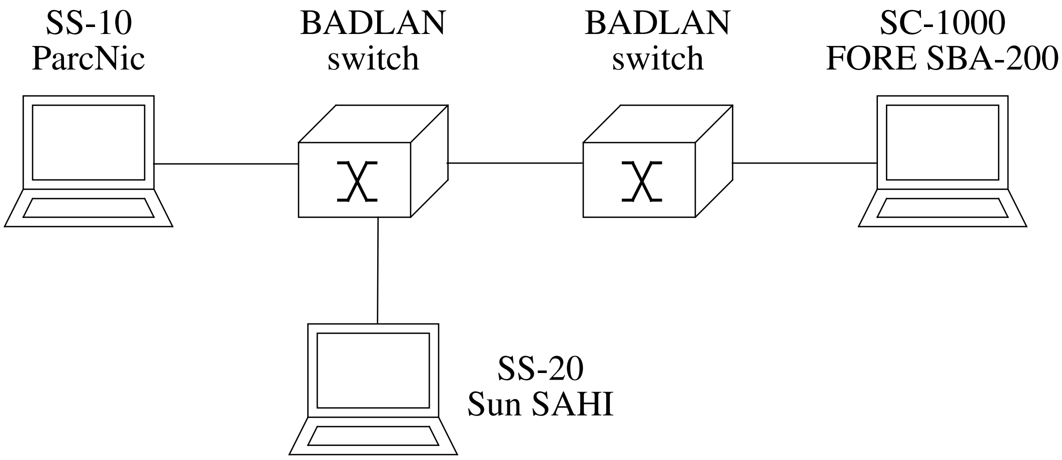

Fig. 6. Test configuration. Two switches (BADLAN) connect a set of workstations (Sparc Stations with a variety of different host adapters)

The primary performance measurement tool used was the `ttcp` program widely available on the net. This programs tests memory-to-memory speed

normalization theory in relational database design.

through TCP or UDP connections. Between the SS10 and the SS20 we measured 32 Mbit/sec using TCP and 45 Mbit/sec using UDP. At this point the machine with our experimental implementation was essentially CPU bound.

It was not surprising that the amount of buffering available for a given stream affects the throughput dramatically. Using `ttcp`'s feature for specifying socket buffer space, we saw a linear increase in throughput up to the value above.

Sending raw cells, the CPU maxes out at about 19,000 cells/sec, or 0.9Mbit/sec. This apparently poor performance must be viewed in the light of the architecture of SunOS network interface drivers, which impose a rather large overhead on each "packet", whether a fully assembled CPCS-PDU presented by the SAR chip or a single cell on a raw channel.

We believe that the limiting factors affecting performance are the SunOS TCP/IP architecture, S-Bus bandwidth and Interrupt latency.

The signaling performance was above expectations. Using Bellcore's Q.port version 1.3 interfaced to our code we measured circuit setup times of about 65 milliseconds per hop. There should be no difficulty meeting the ITU's recommended setup time of 9 seconds.

5 Conclusions

This paper described a prototyping effort in the classical IP (*internet protocol*) and ARP (*address resolution protocol*) over ATM (*asynchronous transfer mode*) model.

It introduced the main features of IP, ARP and ATM, gave a number of arguments why there is a strong push towards integrating these two technologies, and provided an overview of possible scenarios of integration. The remainder of the paper concentrated on the classical IP and ARP over ATM subnetwork model.

The main contributions of this paper are the experiences gained during the design and implementation of the described system, such as the discussion of our mapping resolution model that is based on a finite state machine. Further experiences include the benefits of dynamicly loadable kernel components, the trade-offs involved with combining development and test platforms, the advantages of a clear text debugging interface, and the advantages of graphical user interfaces for configuration and testing.

Acknowledgements

This work was sponsored in part by the US Advanced Research Projects Agency under contract DABT63–92–C–0034 and by a gift from SUN Microsystems to the COAST laboratory.

We would like to thank Bryan Lyles and Carl Hauser for their technical insight and encouragement. Ron Frederick provided valuable help (and no small amount of code) with the driver prototype.

References

[Armitage and Adams, 1993] Armitage, G. J. and Adams, K. M. (1993). Packet Reassembly During Cell Loss. *IEEE Network*, pages 26–34.

[Atkinson, 1994] Atkinson, R. J. (1994). *RFC–1626 Default IP MTU for use over ATM AAL5*. Network Working Group.

[Bellcore, 1995] Bellcore (1995). *Q.port - Portable ATM Signaling Software*. Bellcore, Bell Communications Research, Piscataway, NJ.

[Bertsekas and Gallager, 1992] Bertsekas, D. and Gallager, R. (1992). *Data Networks*. Prentice–Hall, Englewood Cliffs, New Jersey, second edition.

[Biagioni et al., 1993] Biagioni, E., Cooper, E., and Sansom, R. (1993). Designing a Practical ATM LAN. *IEEE Network*, pages 32–39.

[Boudec, 1992] Boudec, J.-Y. L. (1992). The Asynchronous Transfer Mode: A Tutorial. *Computer Networks and ISDN Systems*, 24:279–309.

[Bradley, 1992] Bradley, T. (1992). *RFC–1293 Inverse Address Resolution Protocol*. Network Working Group.

[Chao et al., 1994] Chao, H. J., Ghosal, D., Saha, D., and Tripathi, S. K. (1994). IP on ATM Local Area Networks. *IEEE Communications Magazine*, pages 52–59.

[Cole et al., 1996] Cole, R. G., Shur, D., and Villamizar, C. (1996). *RFC–1932 IP over ATM: A Framework Document*. Network Working Group.

[Comer, 1991a] Comer, D. E. (1991a). *Internetworking with TCP/IP*. Prentice–Hall, Englewood Cliffs, New Jersey, second edition.

[Comer, 1991b] Comer, D. E. (1991b). *Internetworking with TCP/IP*, volume II. Prentice–Hall, Englewood Cliffs, New Jersey.

[Date, 1991] Date, C. J. (1991). *An Introduction to Database Systems*, volume I. Addison–Wesley Publishing Company, Inc., fifth edition.

[Finlayson et al., 1984] Finlayson, R., Mann, T., Mogul, J., and Theimer, M. (1984). *RFC–903 A Reverse Address Resolution Protocol*. Network Working Group.

[Forum, 1994] Forum, A. (1994). *ATM User–Network Interface Specification, Version 3.1*. Prentice–Hall, Englewood Cliffs, New Jersey. Q.2391.

[Forum, 1996] Forum, A. (1996). *ATM User–Network Interface Specification, Version 4.0*. ATM Forum.

[Heinanen, 1993] Heinanen, J. (1993). *RFC–1483 Multiprotocol Encapsulation over ATM Adaptation Layer 5*. Network Working Group.

[Kercheval, 1994] Kercheval, B. (1994). ATMARP: An Architecture Proposal. IP over ATM working group.

[Laubach, 1994] Laubach, M. (1994). *RFC–1577 Classical IP and ARP over ATM*. Network Working Group.

[Leland et al., 1993] Leland, W. E., Taqq, M. S., Willinger, W., and Wilson, D. V. (1993). On the self-similar nature of Ethernet traffic. In *Proceedings of ACM SIGCOMM '93*, pages 183–193. also in *Computer Communication Review* 23 (4), Oct. 1992.

[Mills, 1984] Mills, D. (1984). *RFC–904Exterior Gateway Protocol Formal Specification*. Network Working Group.

[Partridge, 1993] Partridge, C. (1993). *Gigabit Networking*. Addison–Wesley Publishing Company, Inc.

[Plummer, 1982] Plummer, D. C. (1982). *RFC–826 An Ethernet Address Resolution Protocol*. Network Working Group.

[Postel, 1980] Postel, J., editor (1980). *RFC–768 User Datagram Protocol*. Network Information Center.

[Postel, 1981a] Postel, J. (1981a). *RFC–791 Internet Protocol*. Information Science Institute, University of Southern California, CA.

[Postel, 1981b] Postel, J., editor (1981b). *RFC–792 Internet Control Message Protocol*. Information Sciences Institute, USC, CA.

[Postel, 1981c] Postel, J., editor (1981c). *RFC–793 Transmission Control Protocol*. Information Sciences Institute, USC, CA.

[Postel and Reynolds, 1988] Postel, J. and Reynolds, J. K. (1988). *RFC–1042 A Standard for the Transmission of IP Datagrams over IEEE 802 Networks*. Network Working Group.

[Postel and Reynolds, 1994] Postel, J. and Reynolds, J. K. (1994). *RFC–1700 Assigned Numbers.* Network Working Group.

[Suzuki, 1994] Suzuki, T. (1994). ATM Adaptation Layer Protocol. *IEEE Communications Magazine*, pages 80–83.

[Trived, 1982] Trived, K. S. (1982). *Probability & Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice–Hall, Englewood Cliffs, New Jersey.

A From PDUs to Cells

In this section we explain the necessary encapsulations and modifications for the conversion of IP datagrams to ATM cells. Figure A.1 sketches the different stages of this process: LLC encapsulation (sections A.1 and A.2), adaptation layer convergence (section A.3.1), and segmentation and reassembly (section A.3.2). It is important to indicate that we concentrate on the adaptation of the TCP/IP protocol suite to ATM. Multiprotocol encapsulation and convergence are defined for a variety of other protocols and types of applications, which is beyond the scope of this paper.

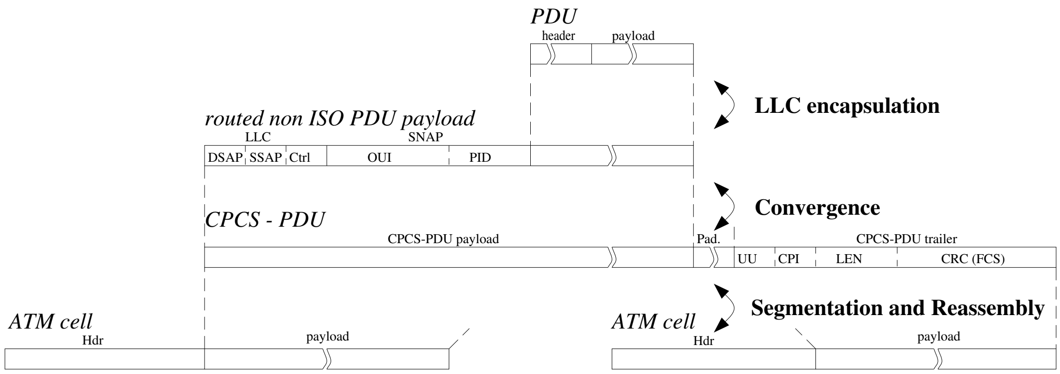

Fig. A.1. The transformation from a network layer PDU to ATM cells

As mentioned above, the basic unit of data transmission in the TCP/IP protocol suite is called a datagram. [Postel, 1981a, section 3.1] defines the contents of datagram headers (20 – 60 octets long) and the possible length of its protocol data unit (up to $2^{16} - 1$ octets incl. header). [Postel, 1981a] also standardizes the fragmentation and reassembly of datagrams within the IP layer.

ATM cells are 53 bytes long. The first 5 bytes are header information. They contain the VC identifier that is used in ATM switches, across *user network interfaces* (UNI), and *network network interfaces* (NNI) for routing. Additionally, cell headers contain a payload type identifier, a priority bit, and a checksum, called *header error control* (HEC). Cells that cross the UNI also contain generic flow control information. The remaining 48 bytes are called the data portion or payload. Cells are rather small compared to IP datagrams. ATM cell layout is defined in [Forum, 1994].

A.1 Multiprotocol Encapsulation

[Heinanen, 1993] describes two methods for carrying routed and bridged *protocol data units* (PDU) over an ATM network. In contrast to [Heinanen, 1993] we will not concentrate on the aspects of bridging. PDUs can be carried over a single ATM VC if they contain additional protocol identifiers. This first approach is called *LLC encapsulation*, because the respective PDUs are prefixed by an IEEE 802.2 LLC header and its possible extensions. The second approach is called *VC based multiplexing*. It does higher–layer protocol multiplexing by using a separate VC for each different protocol.

We chose LLC encapsulation, because we preferred not to establish several different VCs between the same pair of hosts. However, we acknowledge the advantage of VC based multiplexing if the dynamic creation of large numbers of ATM VCs is economical and the applied charging model does not excessively depend on the number of simultaneous VCs.

A.2 LLC Encapsulation

LLC encapsulation applies to different types of data communication services: Type 1, an unacknowledged connectionless mode, Type 2, a connection–oriented mode, and a hybrid Type 3, a semi–reliable mode. We consider network layer protocols that operate over LLC Type 1.

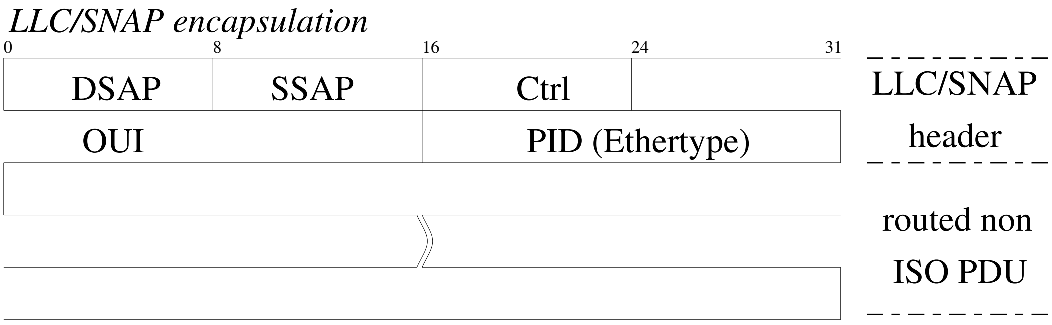

Fig. A.2. LLC/SNAP encapsulation of routed non ISO PDU

The prefixing IEEE 802.2 LLC header consists of three one octet fields. It is possibly followed by an IEEE 802.1a *subnetwork attachment point* (SNAP) header. Figure A.2 depicts the layout of an IP datagram (routed, non ISO PDU) prefixed by its LLC/SNAP header. This format is compatible with the header format specified in [Postel and Reynolds, 1988], the specification of encapsulating IP datagrams and ARP requests and replies on IEEE 802 networks. The value `0x03` in the LLC Control field specifies an unnumbered information command PDU. A LLC header value of `0xAA-AA-03` indicates the presence of a SNAP header. A SNAP header contains 2 fields: A three octet *organizationally unique identifier* (OUI) and a two octet *protocol identifier* (PID). A value of `0x00-00-00` for the OUI specifies that the following PID is an ethertype. Valid values for the PID are defined in [Postel and Reynolds, 1994, pp.168]. We will use only the values for IP datagrams (`0x0800`) and for ARP frames (`0x0806`).

The use of IP and ARP must be only consistent within its LAN type. It is not necessary that it be consistent across all other types of subnetwork technologies.

A.3 The Adaptation Layer

ATM networks are not designed to exclusively transmit connectionless data packets such as IP datagrams. Several kinds of higher level data (such as datagrams, voice samples, or video frames) must be packaged into ATM cells efficiently. This process is referred to as adaptation. It is performed by an adaptation layer that is conceptually located between IP and ATM. It consists of a convergence sublayer, and a segmentation and reassembly sublayer.

A.3.1 Convergence

Four classes of applications were determined by the former CCITT that would require different types of service. *ATM adaptation layers* (AALs) are optimized for their associated class of applications. Depending on which class of application is used, the convergence sublayer performs functions such as multiplexing, cell loss and error detection, or timing recovery. AAL5 is the *ATM adaptation layer 5*, designed for connectionless data applications. It provides the most efficient and functional convergence layer for this class of applications: it is optimized to introduce little header overhead, to require little cell handling cost in host interfaces, and to resemble common data communications interfaces, such as those for Ethernet.

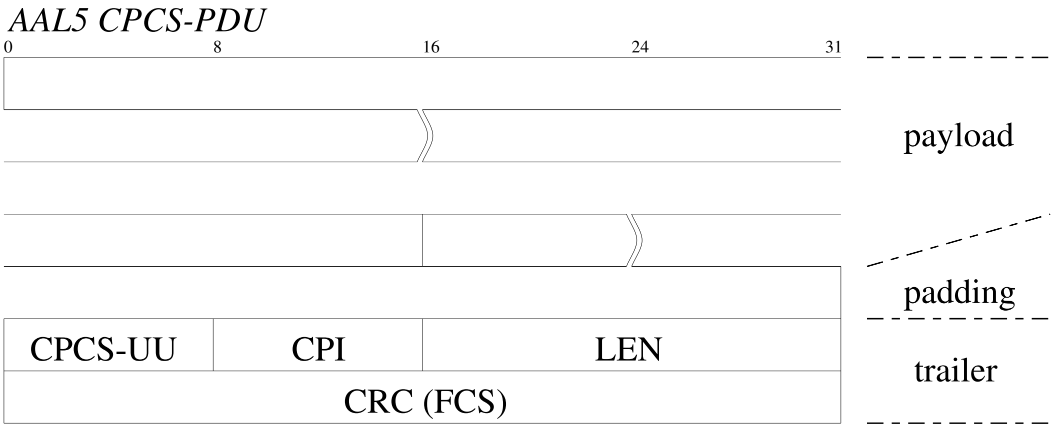

Fig. A.3. AAL5 CPCS–PDU

The convergence sublayer is further subdivided into an upper and lower half, the *service specific convergence sublayer* (SSCS), and the *common part convergence sublayer* (CPCS), respectively. We assume a null SSCS, because the CPCS functions provide sufficient processing for network and transport layers such as TCP/IP.

The AAL5 frame format is specified in [Heinanen, 1993, section 3] and depicted in Figure A.3. AAL5 appends a trailer to the CPCS *protocol data unit* (PDU). See [Atkinson, 1994] for a discussion on the default IP *maximum transfer unit* (MTU) for use over ATM AAL5. The carried PDU is padded such that the total number of octets of the frame is a multiple of the number of octets in the ATM payload (48 octets).

The fields CPCS–UU (*user–to–user* indication) and *common part indicator* (CPI) are both one octet long. Users can transparently transfer user to user information in the CPCS–UU field. It can contain any value. The CPI field has currently no function except for 64–bit alignment of the AAL5 trailer. It contains a null value. The length field determines the length of the payload field in octets. The *cyclic redundancy check* (CRC), also referred to as *frame check sum* (FCS), protects the entire PDU except the CRC field itself.

A.3.2 Segmentation and Reassembly

We mentioned above that the carried PDU is padded by null characters. Let $|name|$ denote the length of entity *name* in octets. The number of octets P required for padding ($P \in [0, |\text{ATM payload}| - 1]$) is determined by formula (A.1).

The *segmentation and reassembly* (SAR) sublayer is responsible for breaking the CPCS–PDU into cells at the sender side and reassembling cells into frames at the receiver side. One of the features of connection–oriented transmission technologies is that data is delivered in sequence. Therefore it is sufficient to flag the final cell of each frame to signal its completion of reassembly. This flag is the user signaling bit, the last bit of the payload type identifier.

$$\begin{aligned} \underline{|\text{CPCS}-\text{PDU payload}| + P + |\text{CPCS}-\text{PDU trailer}|} &= \\ |\text{LLC/SNAP hdr}| + |\text{PDU}| + P + 8 &= \\ 8 + |\text{PDU}| + P + 8 &= k \cdot 48 \\ &= \underline{k \cdot |\text{ATM payload}|} \end{aligned}$$

$$\Longrightarrow \quad P = ((|\text{PDU}| + 16 + 47) \text{ div } 48) \cdot 48 - |\text{PDU}| - 16 \tag{A.1}$$

B Glossary

AAL5	ATM Adaptation Layer 5
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
ATMARP	ATM Address Resolution Protocol
B–ISDN	Broadband Integrated Services Digital Network
CCITT	International Telegraph and Telephone Consultative Committee
COAST	Computer Operations, Audit and Security Technology

CPCS	Common Part Convergence Sublayer
CPCS–UU	CPCS User–to–User indication
CPI	Common Part Indicator
CRC	Cyclic Redundancy Check
CSL	Computer Science Laboratory
CSMA/CD	Carrier Sense Multiple Access w/ Collision Detection
DLL	Data Link Layer
DQDB	Distributed Queue Dual Bus
EGP	Exterior Gateway Protocol
FCS	Frame Check Sum
HCC	Host Call Control
HEC	Header Error Control
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Standards Organization
InARP	Inverse Address Resolution Protocol
InATMARP	Inverse ATM Address Resolution Protocol
LAN	Local Area Network
LATM	Local Area Network ATM
LIS	Logical IP Subnetwork
LLC	Logical Link Control
MAC	Medium Access Control
MTU	Maximum Transfer Unit
NF	Normal Form
NHRP	Next Hop Routing Protocol

NNI	Network Network Interface
OSI	Open Systems Interconnection
OUI	Organizationally Unique Identifier
PARC	Palo Alto Research Center
PARCNIC	PARC Network Interface Card
PDU	Protocol Data Unit
PID	Protocol Identifier
PNNI	Private Network Network Interface
PVC	Permanent Virtual Circuit
QoS	Quality of Service
RARP	Reverse Address Resolution Protocol
ROLC	Routing Over Large Clouds
SAR	Segmentation and Reassembly
SCC	Switch Call Control
SNAP	Subnetwork Attachment Point
SSCS	Service Specific Convergence Sublayer
SVC	Switched Virtual Circuit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UNI	User Network Interface
VC	Virtual Circuit
VCI	Virtual Circuit Identifier
VPI	Virtual Path Identifier
WAN	Wide Area Network
WATM	Wide Area Network ATM