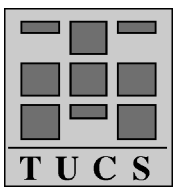
On Communications Protocols and their Characteristics Relevant to Designing Protocol Processing Hardware

Seppo A. Virtanen
University of Turku, Turku Centre for Computer Science,
Lemminkäisenkatu 14, FIN-20520 Turku, Finland



Turku Centre for Computer Science
TUCS Technical Report No 305
September 1999
ISBN 952-12-0533-4
ISSN 1239-1891

Abstract

This report discusses the main functional characteristics of important communications protocols as well as current protocol processing products and research directions in order to determine whether the development of an optimized protocol processing microprocessor should be pursued. The IP, ATM, 802.11 MAC and SDH protocols are discussed with emphasis on the requirements these protocols set on protocol processing hardware.

Keywords: communications protocol, asynchronous transfer mode, synchronous digital hierarchy, internet protocol, medium access layer, microprocessor, general protocol processor

TUCS Research Group
Embedded Systems

Acknowledgements

I would like to thank professor *Axel Jantsch* of the Royal Institute of Technology and professor *Johan Lilius* of the Åbo Akademi University for their comments and input concerning the content of this report.

Contents

Introduction

Applications and operations that require information transfers between two or more devices need a suitable communications facility. For this communication a number of things must be considered, for example how the devices are connected to each other and how they communicate. The most efficient way of connecting the devices is a direct cable connection, but in practice this is most of the time not possible. One of the devices might be connected to a local area computer network that is connected to the internet, and the other one might be a wireless cellular unit. It is clear that the usage of different information transfer mediums presents a need for different sets of rules for communication. These rules are called communications protocols.

Because the communicating devices might be connected through multiple networks and environments, and different networks and environments require different kinds of low-level (transfer-medium-related) communications protocols, it is obvious that the connection between two devices requires fast processing of different types of protocols and in turn the protocol processing requires fast hardware. This need of custom hardware for protocol processing has brought up the general protocol processor (GPP) concept [5]. In this concept, a programmable microprocessor with protocol-optimized design handles the protocol processing. The GPP is used instead of custom multi-chip hardware or a general-purpose microprocessor programmed for protocol processing. The concept is somewhat analogous to the choice between a digital signal processor and custom signal processing hardware.

In this report the main functional characteristics of four important communications protocols are discussed to give an overview of the hardware requirements of protocol processing. The protocols discussed in this report are the internet protocol (IP and IPv6), asynchronous transfer mode (ATM), 802.11 wireless LAN medium access control layer (IEEE 802.11 MAC) and synchronous digital hierarchy (SDH). Then the current protocol processing solutions and products as well as current research directions in the field are discussed.

1. Communications Protocols

Communications protocols are used to transport information from one place to another. The protocols form a stack of layers, each layer communicating with the one above it or below it by passing information in a suitable form to the next layer. The advantage of such a layered system is that each layer abstracts away some technical functions from the layer above it. So e.g. a programmer designing a new application layer program such as a WWW browser does not have to worry about knowing which voltage level in the cable corresponds to a certain logic state in the physical layer.

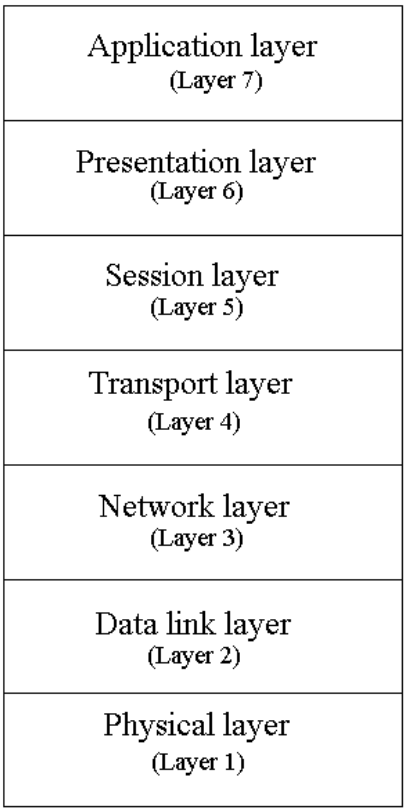


Figure 1. The ISO OSI Reference Model.

An important advantage of this construction is naturally the possibility to use many different physical mediums and well designed custom protocols for each medium type to perform the same high level task. The International Standardization Organization (ISO) has created a protocol layer model to ease protocol design by distinguishing the typical tasks needed in communication. This model is called the open systems interconnection (OSI) reference model [8]. This model is shown in figure 1.

Of the protocols discussed in this report, IP is the highest level protocol, located in the network layer of the protocol stack. ATM and SDH are located in the lower layers of the stack as shown in Figure 2. Sometimes ATM is sent over an SDH connection, in which case SDH would be the lowest level protocol of those discussed here.

1.1. Key Characteristics of Protocol Processing

In order to develop a general protocol processor certain distinct protocol processing operations that vary only a little from one protocol to another must be determined, and functional blocks that can perform these kinds of operations need to be possible to implement into a microprocessor. Jantsch et al. [5] find that there are at least three such operations or qualities in protocol processing: pattern matching and replacement on bitstrings (especially frame or cell header analysis), control dominated operation (large finite state machines and nested if-then-else and case structures) and the need for irregular memory accesses (managing tables and buffers of various sizes). In the following the IP, ATM, 802.11 MAC and SDH protocols are discussed to verify these and to see if there are other such operations common to many communications protocols.

1.2. IP

1.2.1. Introduction

The IP protocol, or the internet protocol, is the network layer protocol used in the internet. It has been designed solely for internetworking purposes. In IP, datagrams (basic IP protocol data units) are sent from a source computer to a destination regardless of their physical locations or the number of subnetworks between them. The following discussion presents version 4 of the internet protocol (IPv4).

A key feature of the internet protocol is unreliable data transfer. This means that the IP protocol itself does not contain functions for notifying the sender of a datagram whether the receiver has successfully received the datagram. A datagram may be discarded at any time anywhere along the transmission path without anyone ever being notified. Because of this, if a reliable connection is needed, the upper layer protocols must provide the notifications and error correction.

A possible protocol stack that utilizes the IP or IPv6 protocol (discussed later) is shown in figure 2. The layer placement of the protocols discussed in this report can also be seen in this figure. Notice that there are two layers missing when the IP protocol stack is compared to the OSI reference model: the presentation and session layers.

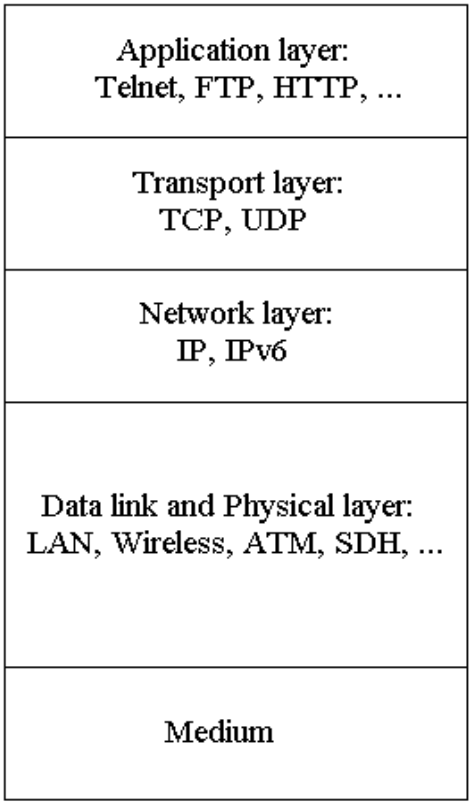


Figure 2. IP / IPv6 protocol stack.

The IP protocol works closely with a transport layer protocol (the TCP protocol, or transport control protocol), to send information from source to destination. The TCP layer breaks up the outgoing information to IP datagrams, and then the IP layer adds its headers to the datagrams and sends them out to the internet. The datagrams may be broken up into smaller pieces as they travel through the internet. On the receiving end, as all the datagram pieces arrive, the IP protocol reassembles the original datagram and moves it to the TCP layer for further processing.

In IP it is possible that the pieces of datagrams and also complete datagrams arrive out of order, so the reassembly process has to buffer incoming datagrams and their fragments until all pieces have arrived. Then the pieces are reordered and put together.

1.2.2. IP Datagram Format

An IP datagram consists of two parts, the header and the text part. The text part is essentially the user data part of a datagram. Both of these can have a variable size, and the total maximum size of a datagram is 64 kB. In practice however, datagram sizes are usually in the range of 1500 bytes. In the header there is a 20-byte fixed-length part followed by a variable length options part.

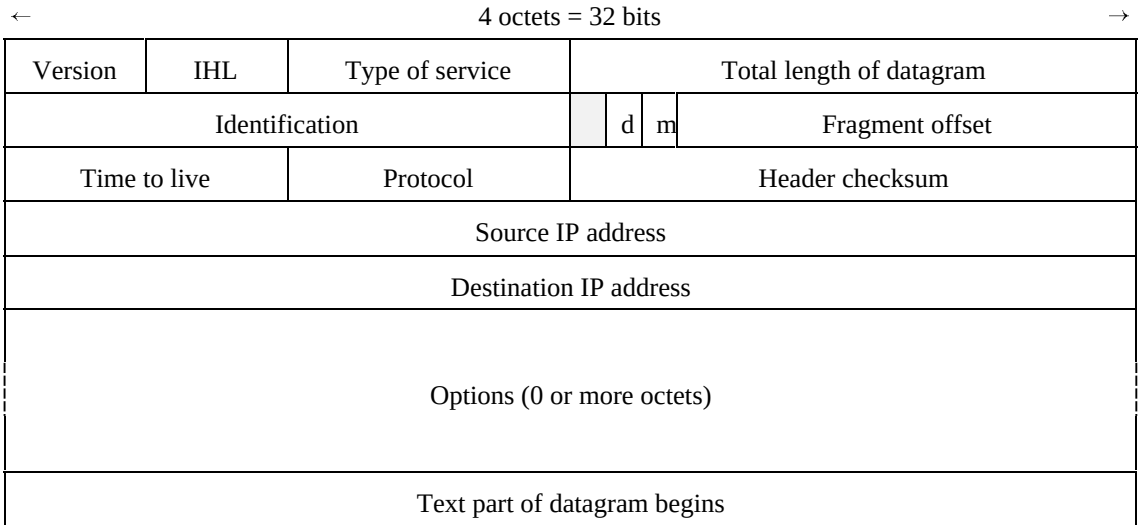


Figure 3. The IP datagram header. IHL = IP datagram header length, d = do not fragment control bit, m = more fragments control bit. Grayed bit is unused [3, 8].

The **Version** field in the datagram header indicates to which version of the IP protocol the datagram belongs to. For IPv4, the value of this field is 4.

The **IHL** field (IP datagram header length) indicates, how many 32-bit data words form the variable-length header of the datagram. This value has to be in the range of 5 to 15.

The **Type of Service** field indicates the speed and reliability requirements for the datagram's transmission. This field is most of the time ignored by routers [8].

The **Total Length of Datagram** field indicates the size of the entire datagram, both the header and the text parts. The maximum value for this field is 65535 bytes.

If the datagram is broken into fragments along the transmission path, then the **Identification** field of each fragment indicates, to which datagram the fragment belongs to.

The two control bits, **DF** and **MF**, are used for controlling fragmentation. If set, the DF bit indicates that the datagram may not be fragmented along the transmission path but must be delivered in one piece. The MF bit indicates that there are still more fragments of the datagram to come. The last fragment does not have this bit set, but all other fragments do.

The **Fragment Offset** field indicates the position in the original datagram where the fragment belongs to.

The **protocol** field is used to indicate which transport layer process takes the datagram after it has been reassembled at the receiving host's network (IP) layer. Possibilities are e.g. TCP and UDP.

The **Time to Live** field indicates the time left in seconds before the datagram must be discarded. Usually this value is decreased at each network hop instead of being decreased every second. This field ensures that a datagram can not wonder around the internet forever.

The **Header Checksum** field is used to verify the datagram header's correctness. Because at least one field in the header changes at each network hop (Time to Live is decreased), this value must be recalculated at each hop. The value is supposed to be zero if everything is in order, and it is calculated by adding each 16-bit subword of the header using one's complement arithmetic, and then by taking one's complement of the result. If the checksum is not correct, then the entire datagram is discarded.

The **Source Address** field contains the IP address of the device that sent the datagram, and the **Destination Address** field contains the IP address of the receiving device.

The **Options** fields can contain values that concern security and routing issues, as well as any user defined options. One purpose of these fields is to allow experimenting without the need for allocating fixed non-used fields to each datagram.

1.2.3. Addressing and Address Resolution

The hosts that are connected via and through the internet are identified by a unique 32-bit number called IP address. Usually the 32-bit address is presented as four 8-bit numbers separated by dots, e.g. 130.232.128.73. The IP address space is divided into four classes named A, B, C, D and E. Of these, each class A network can hold as many as 16 million hosts, each class B network 65 535 hosts and each class C networks 254 hosts. There can be a maximum of 126 class A, 16 382 class B and roughly 2.1 million class C networks. Class D addresses are so called multicast addresses, in which a datagram has multiple receivers, and class E addresses are reserved for future use.

Because IP is a network layer protocol, it is necessary to transform IP addresses into addresses that can be understood and processed by the data link layer. For this transformation (and its inverse, transforming data link layer addresses into IP addresses) address resolution is needed. In the internet two protocols are used for these purposes: the address resolution protocol ARP and the reverse address resolution protocol RARP. A host that wishes to send a datagram to another host broadcasts a query in its local area network, asking for the hardware (data link layer) address of the host with the intended receiver's IP address. If the receiver is in the same LAN it replies with its hardware address (e.g. ethernet address). Otherwise, the IP datagram can for example be sent to a default hardware address (e.g. the LAN router). The host at the default hardware address would then send it to the router of the receiving host's LAN.

RARP is needed for e.g. booting a workstation without a hard disk through the network. As power is turned on, the workstation broadcasts its hardware address, asking for its IP address. If there is a RARP server connected to the LAN, it replies to the broadcast with the workstations IP address.

1.2.4. Conclusions for IP

In terms of required processing in networks running the IP protocol, some key characteristics that make demands on the hardware used can be identified:

◦ ***Datagram header analysis.*** Each IP datagram's header is inspected at every IP router for errors by calculating the header checksum field. The header's length and the entire datagram's length are extracted from the header, as well as the type of the protocol used in the transport layer. These tasks require bitstring matching (a field in an incoming header is compared to possible values in memory and a decision is made based on the matched value).

◦ ***Memory***. Fast data memory is needed to buffer incoming datagram fragments for reassembly and to avoid network congestion and possible discarding of datagrams.

◦ ***Checksum calculation***. Prompt checksum calculation is required because the datagram header checksum must first be verified at each network hop, and then recalculated to match the changed header (at least one header field always changes at each network hop). For this a unit with basic arithmetic functions (summing and one's complement calculation) is needed. Arithmetic functions are also needed for modifying the TTL field in datagram headers.

◦ ***Support for variable-length data units***. The IP datagrams need special processing, because both the header and the text part of a datagram are not fixed in size.

1.3. IPv6

1.3.1. Introduction

The internet has been growing very fast since it became available to the general public in the 1990's. It has rapidly changed from an academic network of information exchange to a commercial communication channel, relaying messages, images, movies, banking services etc. As the number of hosts connected to the internet grows, it is easily foreseeable that at some point in time the 32-bit address space of the current version of IP, IPv4, will become fully utilized. This development and the movement towards mobile wireless communication has created the need for a faster, more secure, more type-of-service oriented protocol that can support many more hosts than IPv4.

After much discussion, a protocol called SIPP, Simple Internet Protocol Plus, was formed based on two next-generation internet protocol proposals presented in 1993. This protocol was then selected to become internet protocol version 6, IPv6. It solves the problem of possibly running out of IP addresses in the future by introducing 16 byte, or 128-bit, addressing to replace IPv4's 4-byte (32-bit) addressing.

Other improvements in IPv6 when compared to IPv4 are improved security and type-of-service features as well as a fixed size datagram header with only 7 fields (IPv4 had a variable size with 13 fields). This change makes the processing of IPv6 header information faster.

1.3.2. IPv6 Datagram Format

Like IPv4, an IPv6 datagram consists of two parts, the header and the payload part. The payload part carries the data that is to be transferred between two hosts. The header has a fixed size of 320 bits, or 40 bytes. As a major difference, the header checksum field has been removed as checksum calculation has been shown to be a serious bottleneck in IPv4. It was decided that error correction should take place in the higher levels of the protocol stack.

As some of the header fields of IPv4 are missing, the designer's of IPv6 have provided the possibility of using so called extension headers. These optional additions to IPv6 headers appear immediately after the fixed header. These extension headers can be used e.g. to provide datagram sizes beyond 64 kB (called jumbograms) for supercomputing applications as well as additional routing, security and fragmentation information.

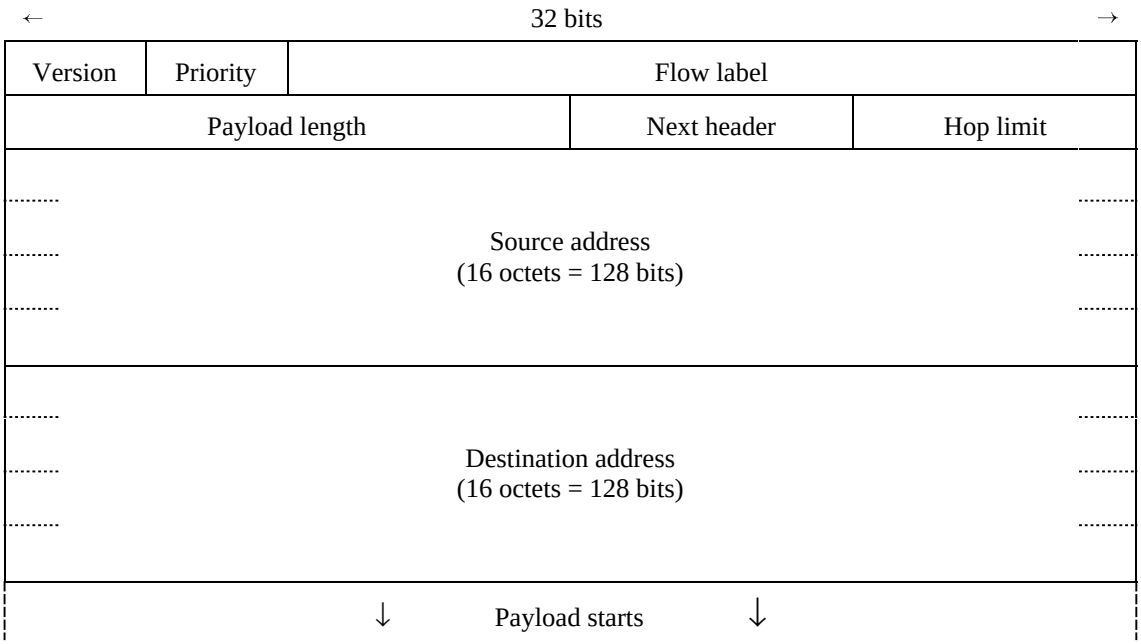


Figure 4. The IPv6 datagram header [8].

The **version** field of the IPv6 header must always contain the value 6 to indicate the protocol version.

The **priority** field is used to indicate the type of service the data requires. The higher the value, the higher the bandwidth requirement. Values from 8 to 15 are intended for real-time services such as audio and video connections. The lower the priority value, the

sooner the packet is discarded if congestion occurs.

Flow labels can be used to setup a connection with special properties between two hosts. It is used to reserve bandwidth between the hosts. If a datagram with a non-zero flow label appears at a router, the router finds out from its tables, what type of special service is needed. The values for flow labels are assigned randomly, and the router lookups are performed by finding an entry with the correct source and destination addresses and the correct flow label.

The **payload length** field indicates how many bytes follow the fixed-length header. This value differs from IPv4's total length value in the sense that the header bytes are not calculated into the payload length value as was done in IPv4.

The **next header** field indicates, what type of extension header follows the fixed length header. If there is no extension header following the main header, then this field indicates the receiving upper level protocol (e.g. TCP, compare to IPv4's protocol field). If the next header field indicates that there is an extension header following the main header, then the payload part of the datagram starts with another next header field. If there are no more extension headers in the datagram, then this field indicates the receiving upper level protocol. Otherwise, this field again indicates the next extension header.

The **hop limit** field is used to prevent a datagram from wandering around the internet forever. The value is decreased by one at each network hop. In practice this field has the same function as the time to live field in IPv4, only this name fits the performed function better.

The **source** and **destination address** fields contain the 128 bit IPv6 addresses of the sender and the receiver of the datagram. Addresses with 80 leading zeros are reserved for the IPv4 address space [8]. The new addresses are written as eight semicolon-separated groups of four digit hexadecimal numbers, e.g. ABCD:1234:7890:0000:0000:000A:0D33:9000. It has been decided that in the notation all leading zeros can be left out and one or more groups of four hexadecimal zeros can be replaced by two semicolons. With these rules in mind the previous IPv6 address could then be written as ABCD:1234:7890::A:D33:9000. IPv4 addresses are expressed by starting the notation by two semicolons, e.g. ::130.232.128.73. Of course IPv4 addresses can also be written as hexadecimal number groups, in which case the address given as an example would become ::82E8:8049. This was formed by first converting each 3-digit decimal number to a binary number, then combining the 8-bit binary numbers into 16-bit numbers and finally converting the resulting 16-bit binary numbers into hexadecimal values.

1.3.3. Conclusions for IPv6

The processing requirements of IPv6 are very similar to those of IPv4. However, the long addresses in IPv6 require higher speeds in address decoding and resolution than in IPv4. The unlimited datagram size places greater demands on buffering and memory. On the other hand, the processing of IPv6 datagram headers is faster than in IPv4 since there are fewer fields to analyze.

1.4. ATM

1.4.1. Introduction

Asynchronous transfer mode, or ATM, is a relatively new technology designed to facilitate future needs of networking. In particular, a goal in designing ATM has been to create a single new network type that could replace the existing telephone system and other specialized networks. It is designed to carry information at very high speeds (155.52 Mbps or 622 Mbps). ATM designers were interested in making a protocol that could be used primarily for transmitting voice and video streams, so in ATM rapid information delivery is more important than accurate delivery [7, 8].

As the name implies, ATM is an asynchronous protocol. This means that ATM packets, called cells, are sent only when there is some information to send. By contrast, in synchronous protocols packets are sent all the time at given intervals. This means that in synchronous transmissions packets are sent even if there is nothing to send. Also, even if the information to be sent would fit into one synchronous packet, it is very often split into two parts because the synchronous packets need to be sent on time and the information arrives to the packet forming layer at random times. It is clear that ATM then does not require as heavy processing for outbound cells as synchronous systems do.

In PSTNs (Public Switched Telephone Networks) a method called circuit switching is used to send information from the sender to the receiver. Basically circuit switching means that a copper wire circuit is formed between the calling parties by means of using switches that connect copper wires to one another. ATM is a packet switching technology with point-to-point connections (there is only one sender and one receiver at the ends of the transmission medium). This means that instead of using switches to form a direct wire connection with the calling parties data cells are sent from the source through a series of point-to-point routers to the destination. The route of the data cells is called a virtual circuit, and the routers are called ATM switches. The virtual circuit is set up for each connection and then terminated after the transfer (e.g. the circuit is set up for the duration of a phone call). This method of transmission has many advantages: first, the ATM cells moving on the same virtual circuit always arrive in the same order they were sent, and thus no cell reordering is required in the receiving end. Second, signal detours around faulty parts of networks can be easily formed by changing the routing information, no wire switching is required.

In ATM networks the connection hierarchy has two levels: virtual circuits and virtual paths. Several virtual circuits form a virtual path similarly as several insulated copper wires would form a cable. The idea of this two level hierarchy is that if due to a network problem rerouting is necessary, then rerouting a single virtual path causes automatically the rerouting of the virtual circuits inside the virtual path. Thus the virtual circuits (single connections) do not need to be rerouted individually, but can be rerouted as a batch.

The ATM specification does not standardize the format for transmitting cells. Cells can be sent independently or they may be embedded into a carrier, e.g. T1, T3 or

SONET/SDH. For the ones given as examples there are standards specifying how ATM cells are packed into the frames provided by these systems.

1.4.2. ATM cells

Each ATM cell contains 53 bytes (each byte is 8 bits). Of this, the ATM header consists of the first 5 bytes and the cell payload consists of the last 48 bytes as shown in figure 5.

Header (5 bytes)	User Information (Payload) (48 bytes)

Figure 5. ATM cell [7, 8].

The ATM header shown in figure 6 contains information on the virtual circuit and virtual path the cell is to be transmitted along. As the path information is a 12 bit integer and the circuit information is a 16 bit integer, it is easily determined that a host could have up to 256 incoming and outgoing virtual paths, each containing 65 536 virtual circuits. Actually the number of virtual circuits available is smaller since some VCI values are reserved for network control operations like setting up connections.

VPI (12 bits)	VCI (16 bits)	PTI (3 bits)	CLP (1 bit)	HEC (8 bits)

Figure 6. ATM cell header (network-to-network interface). VPI = Virtual Path Identifier, VCI = Virtual Circuit Identifier, PTI = Payload Type Identifier, CLP = Cell Loss Priority, HEC = Header Error Check [7, 8].

The payload type identifier (PTI) indicates whether the cell is a user data cell or a maintenance/resource management cell. Also, problems in cell transmission (congestion etc.) are indicated in the payload type identifier.

If there is congestion in an ATM switch and cells need to be discarded to regain normal operation, then cells with cell loss priority (CLP) set to 1 are discarded before cells with CLP=0.

The header error check (HEC) field of the header contains a checksum of the ATM cell header. ATM has error correction for only the cell headers to ensure correct delivery addresses. ATM error correction is discussed later. As ATM is designed on the other hand for use with optical fiber and on the other hand for transmitting video and voice, small errors in the payload are acceptable. Also, the error rates on optical fiber transmissions are very small.

1.4.3. ATM protocol stack and data transfer

As seen in figure 7, the protocol stack is somewhat different in ATM when compared to a traditional protocol stack (figure 2). Actually an ATM network can not be conclusively subdivided into parts fitting the traditional protocol model, since single

layers of the ATM stack perform tasks that are usually distributed over the entire protocol stack. For example, the ATM layer is usually regarded as a data link layer protocol. However, a data link layer protocol is often defined as a single hop protocol used by machines at the opposite ends of a wire, but the ATM layer has actually the characteristics of a network layer protocol: end-to-end virtual circuits, switching and routing [8].

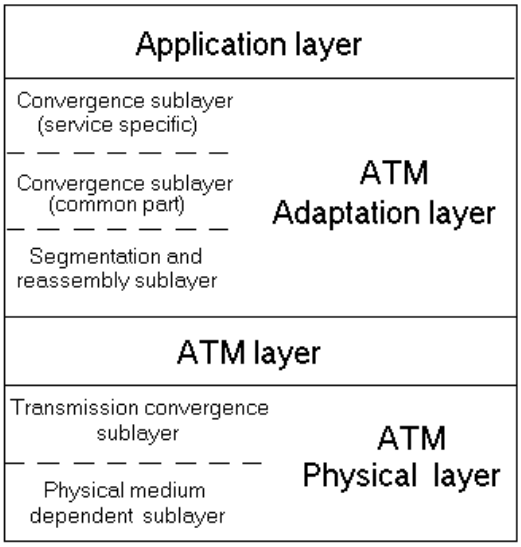


Figure 7. ATM layers and sublayers.

Let us now follow a message sent by an application through an ATM network to an application on another host. The message comes from an application to the convergence sublayer of the ATM adaptation layer (AAL). The convergence sublayer may give the message a header and/or a trailer, depending on the applications needs and the service specific part of the convergence sublayer. Then the message and the headers and trailers added to it are split into data units of 44-48 bytes. The data units are passed to the segmentation and reassembly (SAR) sublayer. The SAR sublayer may add its own header and/or trailer to each data piece and then passes them on to the ATM layer. There is no error or flow control in the ATM layer, but the AAL provides services to application programs and saves them the trouble of splitting data into cells at the source and to reassemble data at the destination.

Upon acquiring the first data unit the ATM layer will start setting up a virtual circuit to the receiving host, if one has not been formed yet. The ATM layer sends a request cell to establish a transfer route. The switches along the transmission path return virtual route information to the unit that requested the route. The route information is then included in the cell header's (VPI and VCI) of all outbound data units. When the virtual circuit is established, the ATM layer just outputs 53 byte cells one after another to the physical layer. The ATM layer does not provide any acknowledgements of receiving remote cells: ATM was designed for use on fiber optic networks, which are highly reliable. The transport medium for ATM is usually fiber optics, but for transmissions of less than 100 meters coaxial cable or category 5 twisted pair can be utilized [2, 7, 8].

ATM cells can be sent through a network directly or via a carrier. The ATM layer provides outwards a sequence of cells, and the transmission convergence (TC) sublayer of the physical layer provides a uniform interface between the physical medium dependent (PMD) sublayer and the ATM layer. The PMD sublayer is the one that is concerned with putting bits in a correct format out to the medium. The PMD encodes the cells from the ATM layer as necessary and pushes them out as a bit stream. The PMD is medium and hardware dependent. On the receiving end, basically the opposite of what was described needs to be done. The main difficulty lies in the synchronization to the incoming bitstream. This is no problem, if the ATM cells are carried by another physical layer protocol that has its own synchronization method. However, if ATM cells

are sent directly on the medium, then some means of synchronization needs to be utilized. For this task, the header error check (HEC) field of the ATM cells is used. The TC layer maintains a 40-bit shift register for the incoming bit stream. The 40 bit value is examined to determine, if it could be a valid ATM header (the last 8 bits form a correct checksum for the first 32 bits). If the last 8 bits do not form a valid HEC, then the register is shifted left one bit, and the next bit in the incoming stream is inserted as the rightmost bit in the register. This is done until a valid HEC is found.

When the register contains a valid ATM cell header (which still could be a random bitstring and not the intended header), the next 424 bits (48 bytes, possible cell payload) are discarded and the 40 bits after them are read into the register. The process is repeated until δ consecutive valid headers are found. This way the possibility of a random bitstring being interpreted as a valid row of ATM cells is $P = 2^{-8\delta}$ [8].

The previously described method works the same way for error correction: after finding δ consecutive valid headers, normal operation starts. In normal operation, if α invalid consecutive headers are encountered, the system goes back to hunting, shifting the incoming bitstream first to find one valid HEC and then to find δ consecutive valid ones.

This process is best represented as a state machine with three states [8]. The bit-by-bit input analysis is done in the HUNT state, and when a correct HEC is found, the system goes into the PRESYNCH state, where cell-by-cell header checking is done to find δ valid consecutive headers. If δ valid consecutive headers are not found, then the system returns to the HUNT state. Otherwise, the system goes into the SYNCH state. Now, if α invalid headers are found, the system returns to the HUNT state. Otherwise, normal cell processing is executed in the SYNCH state.

1.4.4. Conclusions for ATM

ATM is seen as a potential future technology for wired communications, suitable for both telephone systems and data transfer systems [2]. One of the advantages of ATM is a relatively small and fixed transmission unit size, and the increase in transmission speed provided by the minimal error correction. For the needs of voice and video transfers it is much more important to have a high speed transmission path rather than to have a lower speed error-free one. Also, because of the use of optical fiber for ATM transmissions, the error rate is very low despite of the minimal error correction and detection mechanisms. If better error correction is needed, the AAL and the upper layers of the protocol stack can use their own mechanisms.

In terms of required processing in ATM, there are a few characteristics of the protocol that make demands on the hardware used:

◦ ***Cell header analysis.*** Each ATM cell's header is inspected for checksum errors, and also the headers are inspected to extract routing and addressing information. These tasks require bitstring matching (a field in an incoming header is compared to possible values in memory and a decision is made based on the matched value). Also prompt checksum calculation is required, and for this some kind of arithmetic unit is needed.

◦ ***Fast data moves***. Based on the result of bitstring matching, it may be necessary to move an ATM cell's payload quickly to another memory location, another layer in the protocol stack or to a system output. Fast data transfer buses are needed for this purpose.

◦ ***Fast bit-wise shifting***. In the process of receiving an ATM transmission fast bit-by-bit shifting is needed. The incoming bitstream needs to be analyzed using a shifter to determine ATM cell boundaries.

◦ ***Memory***. Fast data memory is needed to buffer cells in ATM switches to avoid congestion and e.g. when trying to synchronize to an ATM bitstream.

◦ ***Fast ATM cell formation and cell decoding***. The information that comes from upper layers in the protocol stack must be split into data chunks of 48 bytes and then a 5 byte header must be added to each data chunk. This can be done reasonably well with standard microcontroller blocks by implementing optimized routines for the purpose: write header to beginning of cell, read 48 bytes of data from memory at pointer position, update pointer to point to the beginning of the next 48 byte sequence, write the 48 bytes to the end of the cell. On the receiving end, the cells need to be similarly decoded from 48 byte chunks to larger (original) data units.

1.5. IEEE 802.11 Wireless LAN Medium Access Layer (MAC)

1.5.1. Introduction

As portable and hand held computers and communication devices have become increasingly popular, the need for connecting them easily to different types of networks has risen. From the user's point of view, the easiest way of connecting such a device to a network is to use a wireless connection. This way the user does not have to connect any cables and can use the mobile unit anywhere within the range of a wireless network base station. Clearly wireless communication brings up new problems in data communication, such as increased noise and interference, overlapping LANs and the increased need for securing the connection.

In the following we discuss the properties of the IEEE 802.11 wireless LAN MAC protocol. The protocol is specified to work at speeds of 1 and 2 Mbps. The wireless medium can be infrared at wavelength 850..950 nm, direct-sequence spread spectrum radio at 2.4 GHz or Frequency-hopping spread spectrum radio at 2.4 GHz [4].

1.5.2. Basic functionality

The IEEE 802.11 wireless LAN MAC protocol implements the CSMA/CA method for medium access. This method differs from the CSMA/CD method used in ethernet LAN's (IEEE 802.3) in the sense that the system focuses on avoiding packet collisions (CA, collision avoidance) instead of on detecting them (CD, collision detection). In CSMA/CA a host senses the transmission medium for activity. If the medium is idle, the host waits a short period of time called interframe space (IFS) and then senses the medium again (there are actually three different IFS times, but for simplicity, they are not discussed in more detail here). If the medium still is idle after IFS, then the host may start transmitting data. If the medium is busy at the initial medium sensing, the host

must back off for a random time before attempting to initialize transmission again.

The units that operate in the same 802.11 wireless LAN form a basic service set (BSS). Each unit and each BSS has its own network address. In 802.11 two modes of operation can coexist: contention operation (DCF, distributed coordination function) and contention-free operation (PCF, point coordination function). The contention-free operation is optional and requires one host in the network to act as a point coordinator. The contention-free operation is a transmission mode in which hosts in the same LAN do not compete of network access but transmit data in turns after being polled by the point coordinator. This access method guarantees that each client always gets a certain amount of bandwidth [1].

In contention operation hosts compete of network access by sensing the media as explained above in CSMA/CA. If both are in use, the contention mode and the contention-free mode co-exist in the network so that one mode is always followed by another. The time allocated to each mode changes dynamically based on the hosts' need for each type of traffic in the LAN. The interframe spaces for hosts operating in different modes are defined in such a way that hosts that operate in contention-free mode have a shorter back-off period than hosts operating in contention mode. This way hosts that are operating in contention mode are not able to interrupt the contention-free operation.

1.5.3. Frame Format

Each 802.11 MAC layer frame consists of the following parts [4]:

- A **MAC header** that contains frame control, duration, address and sequence information. The length of the header is 30 bytes (240 bits). The header is studied in more detail in figure.
- A variable-length **frame body** with the actual payload to be carried. The frame body length must be between 0 and 2312 bytes.
- A **frame check sequence** (FCS) that contains a CRC-32 checksum for all the fields of the MAC header and the frame body. The length of the FCS part is 4 bytes. The FCS calculation and the polynomials used for it are defined in [4].

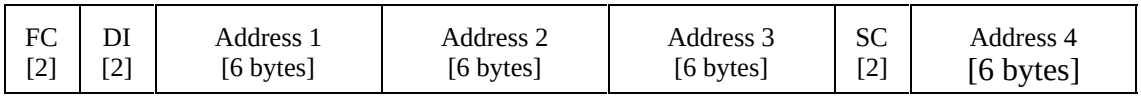


Figure 8. 802.11 MAC header fields (field byte lengths in brackets).
FC = Frame Control field, DI = Duration / ID field, SC = Sequence Control field.

The **Frame control** field of a 802.11 MAC frame includes subfields for protocol version, frame type and subtype, fragmentation control and power management. The protocol version is intended to be changed from 0 to another value in case of a major

change in the protocol specification. The type and subtype subfields indicate the frame's function, i.e. whether the frame is a data, control or management frame and what its key function is (e.g. association/disassociation request or response, beacon, powersave etc.).

In DCF operation the **Duration/ID** field holds a frame type dependent frame duration value (1..32767). If the system is in PCF operation, then this field holds the fixed value 32768. Other values are either reserved for future use or used for carrying association information in that type of frames.

Depending on the type of frame, the 48-bit **Address** fields can carry any of the following: a BSS identifier (the 48-bit address identifying the basic service set in which the host operates), a destination address, a source address, a transmitter address and a receiver address. The format of the 48-bit addresses is the same as in e.g. ethernet LANs and is defined in IEEE standard 802-1990.

The **Sequence Control** field has two subfields, the **fragment number** subfield (4 bits) and the **sequence number** subfield (12 bits). If service or management data needs to be fragmented, then the sequence number subfield contains a specific identification value for the service or management unit in question. Fragments are numbered from zero onwards and the first fragment has the value zero in its fragment number subfield.

1.5.4. Conclusions for IEEE 802.11 MAC

In terms of processing the IEEE 802.11 MAC protocol, the following operations place demands on the hardware used:

◦ ***Timers.*** To facilitate the different interframe spaces needed for different types of operation (DCF and PCF) and also to control back-off timing, real-time timers are needed.

◦ ***Random number generation.*** To be able to give each host in a BSS an equal opportunity to send and receive frames, the stations must back off for a random time period upon detecting traffic on the transmission medium. It is very important that the random back-off periods are statistically random to guarantee access for each station.

◦ ***CRC calculation.*** Every single 802.11 MAC frame sent contains a CRC-32 checksum of the entire frame. Thus, for each outgoing frame, the checksum needs to be calculated and inserted into the frame, and for each incoming frame, the checksum needs to be recalculated in order to detect errors in the transmission.

◦ ***Frame header analysis.*** All Frame headers are inspected for frame type, CRC checksums, fragmentation and addressing information. These tasks require bitstring matching (a field in an incoming header is compared to possible values in memory and a decision is made based on the matched value). As the frame headers and frames vary in size depending on frame types, efficient header processing is very important in terms of overall performance.

◦ ***Fast data buses***. Based on the result of bitstring matching, it may be necessary to move a frame's payload quickly to another memory location, another layer in the protocol

stack or to a system output. Fast data transfer buses are needed for this purpose. Also additional memory for buffering frame fragments is needed.

1.6. SDH

1.6.1. Introduction

The 1980's was a time of major changes in the telephony markets in North America. The Bell telephone company was broken up to small local telephone companies, and AT&T lost its monopoly as a long distance carrier due to new legislation. As the number of long distance telephone operators increased and most of them utilized their own optical TDM systems to connect customers from local telephone companies to long distance networks, the need for a new transmission standard became evident. In addition, the existing wide area transmission standards were from the 1960's and early 1970's and were based on the introduction of digital signal transmission technology for coaxial cable.

Bellcore in the U.S.A. started working on a new standard called SONET (synchronous optical network) for high capacity digital transmission that would be implemented on optical fiber lines. In the course of the development the CCITT (nowadays known as ITU) joined the design effort. The co-operation resulted in the ANSI SONET standard and a set of parallel CCITT recommendations that were called SDH (synchronous digital hierarchy) [8].

The designers of SONET and SDH had four major goals in their work. First, the new standards had to make it possible for different carriers to interwork. Second, some means were needed to unify American, European and Japanese digital telephone carrier systems: all of them were based on 64 kbps PCM channels but all of them combined the channels in different ways. Third, a method for multiplexing multiple digital channels was needed. The system was decided to be based on TDM with the entire bandwidth devoted to one channel that contains time slots for the various subchannels. Fourth, the new standards had to provide support for operations, administration and maintenance (OAM). These weren't well enough supported in existing systems in the 1980's.

The differences between SONET and SDH are minor, although because of them only a subset of SDH is compatible with SONET and the other way round. So with certain options, communication and information transfer between SONET and SDH networks is possible. At this point it is important to point out, that the fourth goal of the standard design process, OAM, is not supported between SONET and SDH networks, and thus a SONET network can not be administered directly from an SDH network and the other way round.

The transmission speeds for the networks are similar, although the basic transmission unit of SDH is sent at 155.52 Mbps (called STM-1, synchronous transport module level one) and the basic transmission unit of SONET is sent at 51.84 Mbps (called STS-1, synchronous transport signal level one). Actually, the STM-1 is three signals similar to the SONET STS-1 multiplexed together, and thus the SONET STS-3 is the equivalent

of the SDH STM-1. The reason for STM-1 and STS-3 having the speed of 155.52 Mbps and for STS-12 and STM-4 having 622.08 Mbps is fairly obvious: these are the ATM transmission speeds, so ATM can be sent using SONET or SDH trunk networks. The standards define SONET and SDH transmission speeds up to 2.4 Gbps.

1.6.2. SDH protocol stack

SDH is a physical layer protocol that is used in long distance telephone systems. In addition to carrying telephone calls, SDH can also be used to carry for example ATM cells.

SDH systems are formed of switches, multiplexers and repeaters. These are all interconnected with optical fiber. An optical fiber going from one device directly to another is called a **section**. A **line** is a run from one multiplexer to another (with possibly one or more repeaters in between). A **path** is a connection between the source and the destination.

Because SDH is synchronous, frames are output every 125 µs (at STM-1) even if there is no data to send. It is easily observed that this 8000 frames per second matches the sampling rate (8 kHz) of PCM telephone systems. An SDH frame consists of 810 bytes, each byte being 8 bits.

SDH is divided into four sublayers as shown in figure 9 [3, 8]. The **photonic** sublayer deals with the physical properties of light and the fiber used for data transfers. The **section** sublayer handles single point-to-point fiber runs. It generates a standard SDH frame at one end and processes it at the other end. Sections can also start and end at repeaters, which only amplify and regenerate the signal but do not change or process it in any way. The section sublayer also provides error monitoring and scrambling (to avoid long all-zero and all-one transmission units that might be interpreted as a faulty transmission line). The **line** sublayer multiplexes multiple tributaries (information streams from e.g. a local telephone company that have a lower bit rate than SDH) onto a single line and demultiplexes them at the other end. This layer also provides synchronization needed for transmissions and deals with the number of lines to be multiplexed and the way they are multiplexed. The **path** sublayer deals with end-to-end (source to destination) issues, like inserting and removing user payloads to and from different information sources into or out of the SDH transmission system.

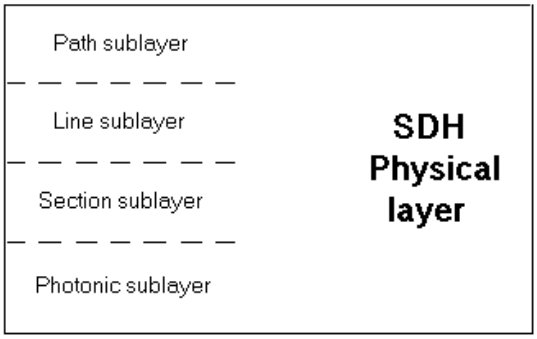


Figure 9. SDH sublayers.

1.6.3. SDH frame format

SDH data is structured into units called frames, and an SDH frame is sent every 125 µs. A frame is divided into nine equal length segments, as shown in figure 10. Each

segment has its own header called the segment overhead.

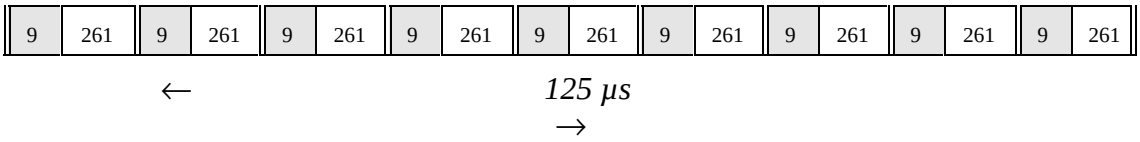


Figure 10. *SDH frame (STM-1). Segments are separated with double lines. Grayed subsegments are segment headers (overheads), and the rest contain the segment payload. Numbers show how many bytes each frame part contains.*

Since the segment overhead bytes are related to one another (i.e. together they form the header information for the entire frame), SDH frames are often illustrated as a segment stack with one segment on top of another. Such a representation of an SDH frame is shown in figure 11. It should be realized that in SDH there is no single frame header, but the information is distributed into the segment overheads.

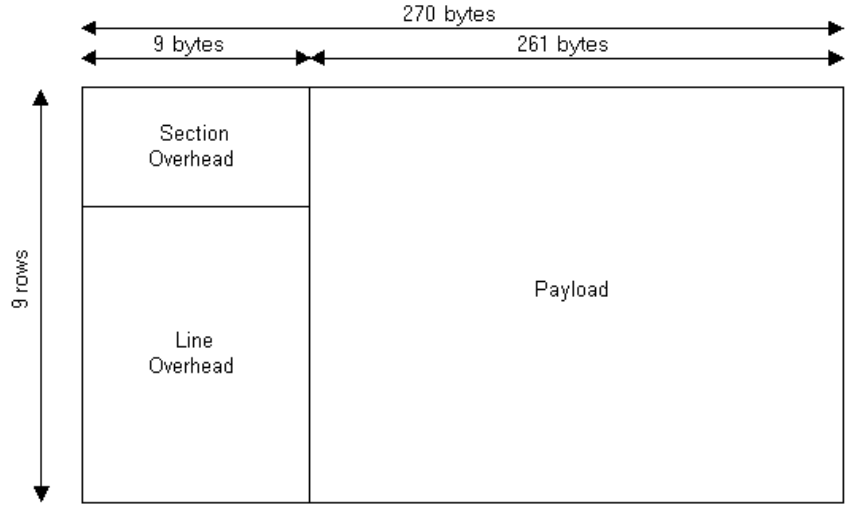


Figure 11. The parts of an SDH STM-1 frame.

The payload part of the frame holds virtual containers (VC's) or parts of them that each carry information from different sources. Each VC starts with a path overhead which contains necessary information for e.g. bit error rate monitoring. A virtual container is virtual in the sense that it is not a sequential burst of data and also because it is not limited by the bounds of the SDH frame. A VC with its path overhead can start anywhere in the payload. Figure 12 shows an example of virtual containers scattered over multiple SDH frames. The starting points and sizes of the virtual containers are defined by the VC's line overhead bytes. The largest VC in an SDH frame is called an administrative unit (AU).

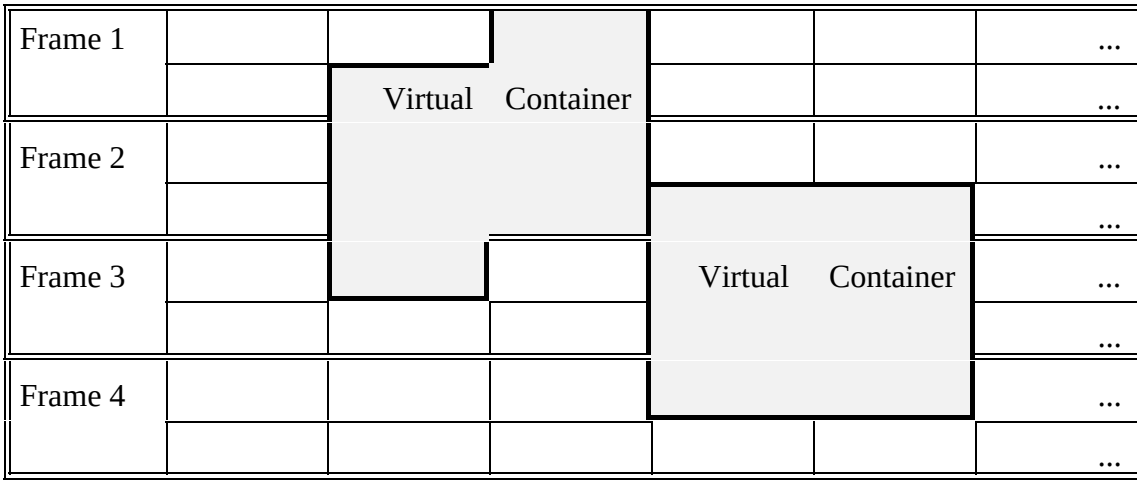


Figure 12. Multiple virtual containers in multiple SDH frames.

Figure 13 gives a closer look at the SDH frame overheads. The section and line overheads are always located in the first nine columns of the segment overhead stack. The path overhead for a VC in the frame is also shown. It is important to notice that not all the bytes in the overheads need to be used. For example, the bytes reserved for voice connections between maintenance personnel and also the bytes used for remotely administering a device on the transmission way are used only when necessary. The abbreviations used in figure 13 are described in table 1 on the next page.

Path overhead for a virtual circuit ↓

Section over-heads	1	A1	A1	A1	A2	A2	A2	C1			
	2	B1			E1			F1			J1
	3	D1			D2			D3			B3
Line over-heads	4	AU Pointer									C2
	5	B2	B2	B2	K1			K2			G1
	6	D4			D5			D6			F2
	7	D7			D8			D9			H4
	8	D10			D11			D12			Z3
	9	Z1	Z1	Z1	Z2	Z2	Z2	E2			Z4
(next frame)											Z5

Figure 13. SDH segment header stack (STM-1) [3].

Table 1. SDH overhead bytes.

A1-A2	Always the first bytes transmitted, used for framing.
B1	An 8-bit parity check for monitoring bit errors in the section.
C1	Identifies a specific STM-1 frame in a higher order (STM-n) frame.
D1-D3	Form a data communication channel for network management messages relating to the section.
AU pointer	Defines the location of the frame's administrative unit within the payload.
E1	Used for orderwire channels (voice channels used by maintenance personnel) in the section.
F1	User channels, available for the management of customer premises equipment.
B2	An 8-bit parity check for monitoring bit errors in the line.
D4-D12	Form a data communication channel for network management messages relating to the line.
E2	Orderwire channels relating to the line.
K1-K2	Form a signaling channel for automatic protection switching of the complete line.
Z1-Z2	Reserved for national use.
J1	Verifies the VC path connection.
B3	An 8-bit parity check for monitoring bit errors in the path.
C2	Indicates the composition of the VC payload.
G1	Used by the receiver to return the status of the received signal back to the transmitter.
F2	Provides a user data communication channel.
H4	Indicates whether the payload is part of a multiframe.
Z3-Z5	Reserved for national use.

1.6.4. Conclusions for SDH

Most of the new optical fiber wide area transmission systems being installed in public networks nowadays utilize either SONET or SDH. SONET and SDH are expected to dominate the market for at least the next decade. Also, it is seen that as SONET and SDH microchips or chips that can be programmed to efficiently run them become cheaper, companies may start using leased optical lines from telephone companies to

connect their different locales to one another or perhaps use such lines for company-wide internet access. Using such high speed optical lines cost-effectively for these purposes would then be made possible by SONET and SDH.

In terms of required processing in SONET and SDH systems, the following distinct characteristics make great demands on the hardware used:

◦ ***Processing speed.*** SONET and SDH are very high speed wide area network systems, the transmission speeds of which currently vary from 55 Mbps to 2.4 Gbps. Because of these high speeds the hardware used for operating these networks must be able to process information at least at 2.4 Gbps.

◦ ***Segment overhead analysis.*** The control and administration information in SDH systems is conveyed as circuit, line and path overhead bytes. The line and path overheads are not transmitted sequentially, but are distributed in 9-octet series that appear every 270 octets in a 2430-octet frame. The path overheads appear as single units every 270 octets. Because of this distributed nature, the system processing an SDH frame must know what type of information is to be expected at the beginning of each segment (270-octet part) in the frame, and also what to do when the path overhead bytes are encountered. So the system must have a logic unit for managing the different segment overheads, and it needs to be able to perform fast bitstring matching (needed for bit error rate monitoring and overhead analysis). Also prompt checksum calculation is required for bit error rate monitoring and descrambling, and for this some kind of arithmetic unit is needed.

◦ ***Fast data moves***. Based on speed of SONET and SDH networks, it is obvious that fast data transfer buses are needed to keep up with the incoming or outgoing data flow.

◦ ***Exact timing***. As it is a synchronous and time-division multiplexed transmission system, SDH requires very exact timing in sending, receiving, assembling and disassembling frames.

◦ ***Fast SDH frame formation and decoding***. The need for fast frame forming comes from the fact that in SDH, STM-1 frames (2430 bytes) are sent every 125 µs. Before sending a frame it must be formed, and upon receiving a frame it must be disassembled.

1.7. Conclusions

In the previous pages an overview of four important communications protocols was presented. From this we can summarize that all the protocols that were examined require certain common operations in the different phases of processing. The necessary device blocks to perform these operations are listed in table 2. In addition to the operations listed in table, processing of these protocols also requires fast memory and fast data buses to facilitate the usage and maintenance of several tables and buffers that are not fixed in size. Also, protocol processing is control oriented, which in practical terms means the usage of many nested case and if-then structures [5].

Table 2. Summary of essential protocol processing functional blocks.

Function	Description
ERROR CHECKER	A unit that makes CRC and other (e.g. IP header checksums) error checking calculations. Programmable, e.g. CRC polynomials can be changed. The Error Checker also calculates checksums for outgoing protocol data units.
RANDOM NUMBER GENERATOR	A unit that generates statistically independent pseudo-random bitstrings. Can produce a random value of any word size.
TIMER	A unit that constantly updates a system clock and several counters. Programmable. The counters can be reset, set to certain values, incremented and decremented either directly or at certain time periods.
SCALER	A unit that handles bitwise shifting and scaling as well as the changing of word lengths. Also takes care of synchronization with the incoming bitstream.
PACKET ASSEMBLER/ DISASSEMBLER	A unit that assembles and disassembles protocol data units. In the assembly process, the outgoing data may need to be split into multiple parts and each part needs to be given a header and possibly a trailer. In the disassembly process headers and possible trailers are removed from the protocol data units and the original data is reconstructed. The incoming data units may need to be reordered in the process.
STATE MACHINE CONTROLLER	A flow control unit that takes care of message passing etc. between protocol operation state machines
MATCHER	A unit that matches bitstrings in the incoming data to bitstrings in the memory, and performs certain actions based on the result of the matching. This unit is the one that analyzes the headers in incoming protocol data units. When sending protocol data units, the Matcher provides some of the information required to construct a protocol data unit header.

To be able to design a well-performing protocol processing device such as the proposed general protocol processor, some of the operations mentioned in table 2 should be implemented in a manner that would outperform solutions implemented on general purpose microprocessors with the same clock frequency. The design should also be optimized for control-oriented processing, and it should take into account the need for very fast and irregular memory accesses. Figure 14 presents a suggestion for the relations between different functional blocks in a general protocol processor.

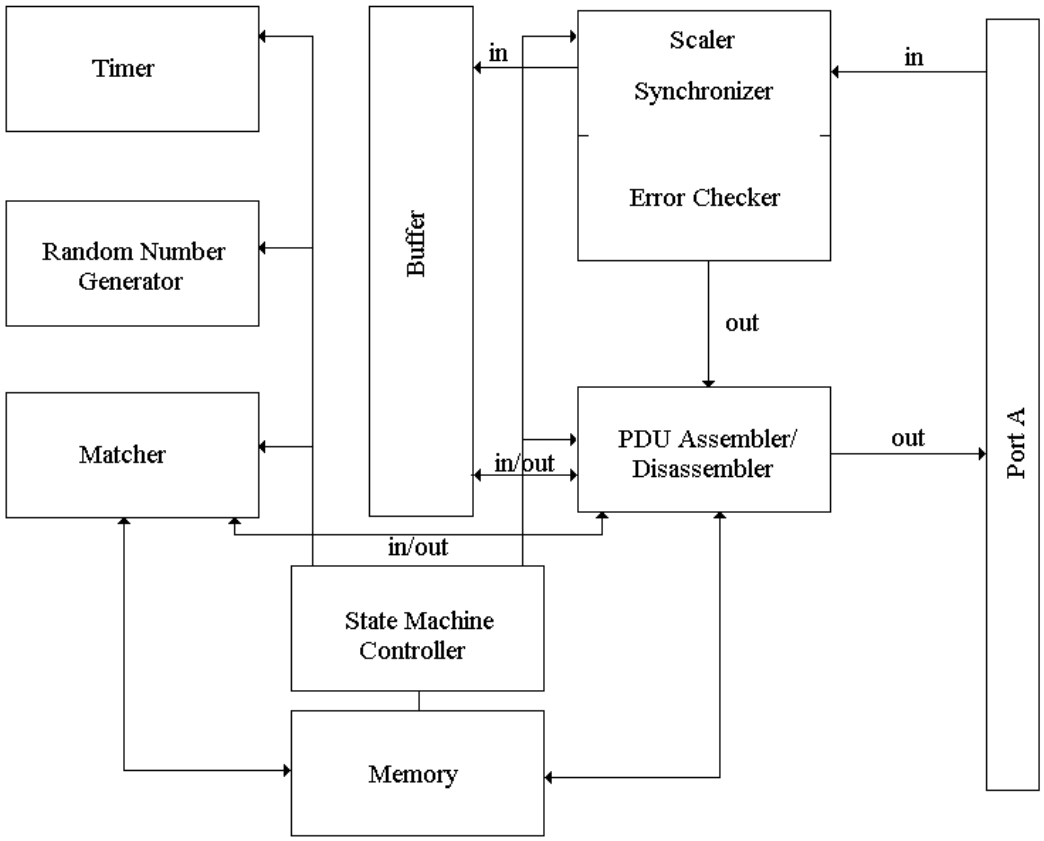


Figure 14. Possible relations between functional units in a general protocol processor.

2. State of Research and Current Products

2.1. Universities

The GPP concept was introduced by Jantsch & al. [5] of the Royal Institute of Technology, Stockholm, Sweden. The research directions covered in their paper are not presented here. The research topics and directions presented here were acquired from the WWW sites of university laboratories and institutes and were either visited directly or found using a search engine.

IMEC [10] carries out research in the areas of wireless systems and digital broadband transceivers among others. The WWW pages reveal some protocol processing related research topics, for example a segment protocol processor for the ATM adaptation layer (I. Bolsens, design methodologies group) and real-time multi-tasking DSP's with some protocol stack control ability (G. Goossens, embedded microcode synthesis group), but there was no mention of any GPP research.

At the University of Arizona, the Advanced protocol design group [15] is doing research to improve existing protocols, especially protocols that include congestion control routines. They have improved the TCP protocol to TCP Vegas, which is much more efficient than the current version of TCP in avoiding network congestion. In a project called Scout [16] they investigate the path concept, which is essentially an extension of a network connection into the operating system. This way the network connection (as well as other functional blocks of the system) is in direct virtual contact with the operating system. Because of this specialized system path that handles network connections directly the system becomes faster.

The MIT Exokernel project [11] aims to replace the multiple high-level abstractions modern operating systems are built on with a kernel that concentrates on multiplexing the hardware efficiently and securely. With this approach, each network application would have an optimal protocol stack coded into it instead of unnecessarily going through each layer in the OSI protocol stack for each task to be performed.

General purpose microprocessor and multiprocessor systems are also researched. An example of such systems is the Stanford Flash multiprocessor [6], which is a general purpose multiprocessor computer system. Many of these projects have a protocol processing application made as a case study, but none of them aim to develop a general protocol processor.

2.1.1. Summary of academic research projects

In conclusion, no general protocol processor research projects were found. The general direction of research is more in developing better protocols as well as system architectures that make it possible to process protocols faster. Table 3 lists the research projects found related to protocol processing.

Table 3. Academic protocol processing related research projects.

Project / laboratory	Type of Research	
IMEC	Hardware:	AAL segment protocol processor, DSP with some protocol stack control ability
Univ. of Arizona: TCP Vegas	Software:	Improving the TCP protocol to prevent network congestion.
Univ. of Arizona: Scout	Hardware:	A new architecture for a more direct interaction between network and operating system
MIT: Exokernel	Software:	Each networking task gets its own optimized protocol stack
Stanford: Flash	Hardware:	A general purpose multiprocessor system with a protocol processing application made as a case study

2.2. Industry

In the telecommunications industry the speed of development is very fast and the newest and most innovative product ideas are not published to general audiences in order to avoid competition and plagiarism. That's why it is difficult to truly assess the industry's research and development situation in the area of protocol processing and more precisely in the possible pursuit towards a general protocol processor.

The assessment is based on information generally available in the World Wide Web. It revealed that many manufacturers are making processors or hardware for processing particular protocols. Such processors are for example the Multispan Communications Protocol Processor [12], the U.S. Software USNET TCP/IP implementation of the Motorola 68EN302 controller [14], the Seiko/iReady TCP/IP iChip [13] and the XaQti GigaPOWER processor [18]. All of these processors are preprogrammed to process certain communications protocols, but are not very programmable for different protocols and environments.

The Multispan processor has built-in support for ISDN, ATM and a few other protocols. The USNET processor combines a microcontroller with a variety of protocols used in the internet and also includes an IEEE 802.3 ethernet controller. The iChip features a TCP/IP stack and an interface to a microprocessor. It is aimed to be used in applications that until now do not utilize communication. Such an application could be e.g. connecting a vending machine to a warehouse for notification purposes (the vending machine runs out of a product). The GigaPOWER processor is an active flow processor with an integrated ethernet MAC unit. It is designed for packet processing as well as for switches and routers that need to provide quality of service functions for the IP protocol.

A recently founded company called Agere has introduced a chip called the Fast Pattern

Processor [9]. Its architecture is optimized for networking applications such as routing, switching and network management. The processor is programmable to support network upgrades via software changes instead of changing the hardware. The processor supports the protocols needed for internet operation and features a high-level programming language. The language is designed specifically for protocol processing.

An interesting view of future needs of networking is provided by Intel. They suggest a virtual interface (VI) architecture [17], in which certain layers of the protocol stack are bypassed for certain types of connections to remove software overhead from transfers. They claim that as the networking hardware becomes faster all the time, the legacy protocol software (e.g. TCP/IP) needed to operate the network is becoming an immensely tighter bottleneck. So their view is to improve the networking software, because they see that it is far behind the current technology level of networking hardware.

2.2.1. Summary of industrial products

In conclusion, there were no general protocol processor products available from hardware manufacturers, and none of the manufacturers revealed having GPP in development. Of the products found, the Agere processor has the closest resemblance to the general protocol processor concept. Also, many manufacturers do make processors that are preprogrammed to process a certain protocol or a few certain protocols and possibly perform general purpose microprocessor functions. Table 4 on the next page lists the industrial products found related to protocol processing.

Table 4. Summary of industrial products covered.

Product	Preprogrammed protocols	Microcontroller or Microprocessor	Other features
Seiko/iReady iChip	TCP/IP stack	No, but a built-in interface to a microprocessor	-
USNET TCP/IP 68EN302	TCP/IP stack and related internet protocols	Yes (Motorola 68EN302).	built-in 802.3 controller and DRAM controller
Multispan protocol processor	Multiple protocols (e.g. ISDN, ATM)	Yes (Motorola 68360).	.
XaQti GigaPOWER	TCP/IP stack and related internet protocols, AppleTalk, IPX, VLAN etc.	-	built-in 802.3 MAC unit
Agere Fast Pattern Processor	Internet Protocols, ATM, AAL5	No (but itself is a programmable processor)	New architecture, optimized programming language
Intel VI	protocol and protocol processing SW redesigned	-	Uses existing networking hardware

References

[1] N. Cravotta, Wireless Standards Vie for your App, EDN Europe May 1999, 55-66 (1999).

[2] M. Decina and V. Trecordi, Convergence of Telecommunications and Computing to Networking Models for Integrated Services and Applications, Proceedings of the IEEE **85**, 1887-1914 (1997).

[3] F. Halsall, Data Communications, Computer Networks and Open Systems (Fourth Edition). Addison-Wesley Publishing Company Inc., USA, 1996.

[4] IEEE Std 802.11-1997, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. The Institute of Electrical and Electronics Engineers, Inc., New York, USA 1997.

[5] A. Jantsch, J. Öberg and A. Hemani, Is there a Niche for a General Protocol Processor Core?, Proceedings of the 16th IEEE NORCHIP Conference, 93-100, Lund, Sweden (1998).

[6] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum and J. Hennessy, The Stanford FLASH Multiprocessor, Proceedings of the 21st International Symposium on Computer Architecture, 302-313 (1994).

[7] R. Onvural, Asynchronous Transfer Mode Networks: Performance Issues. Artech House, Inc., Norwood, MA, USA, 1994.

[8] A. Tanenbaum, Computer Networks (Third Edition). Prentice-Hall, Inc.,Upper Saddle River, NJ, USA, 1996.

[9] Agere Fast Pattern Processor
http://www.agere.com/news/releases/9910.htm

[10] IMEC WWW site
http://www.imec.be/

[11] The MIT Exokernel project WWW site
http://www.pdos.lcs.mit.edu/exo.html

[12] The Multispan Communications Protocol Processor WWW page
http://www.voiceboard.com/cpptech.htm

[13] The Seiko/iReady TCP/IP chip
http://www.techweb.com/wire/story/TWB19990527S0002
http://www.iready.com/

[14] The U.S. Software USNET TCP/IP implementation of Motorola 68EN302 http://netpeer.com/press_releases/1997/08/19/

[15] University of Arizona Advanced Protocol Design WWW site http://www.cs.arizona.edu/protocols/

[16] University of Arizona, Scout project WWW site http://www.cs.arizona.edu/scout/

[17] The Virtual Interface Architecture WWW site http://www.intel.com/procs/servers/segments/vi/index.htm

[18] The XaQti GigaPOWER processor http://www.xaqti.com/gigapower_fs.htm

Appendix A: Acronyms and Abbreviations

AAL	ATM Adaptation Layer
ANSI	American National Standards Institute
ARP	Address Resolution Protocol (in IP)
ATM	Asynchronous Transfer Mode
BSS	Basic Service Set (in 802.11 MAC)
CCITT	Consultative Committee for International Telegraph and Telephony
CRC	Cyclic Redundancy Check
CSMA	Carrier-Sense Multiple-Access
CSMA/CA	CSMA with Collision Avoidance
CSMA/CD	CSMA with Collision Detection
DCF	Distributed Coordination Function (in 802.11 MAC)
DF	Do Not Fragment (in IP)
FCS	Frame Check Sequence (in 802.11 MAC)
FTP	File Transfer Protocol
Gbps	10^9 bits per second
GPP	General Protocol Processor
HEC	Header Error Correction checksum (in ATM)
HTTP	HyperText Transfer Protocol
IEEE	The Institute of Electrical and Electronics Engineers, Inc.
IFS	Interframe Space (in 802.11 MAC)
IHL	IP Datagram Header Length (in IP)
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Standardization Organization
ITU	International Telecommunications Union
kB	10^3 bytes
kbps	10^3 bits per second
LAN	Local Area Network
MAC	Medium Access Control Layer
MF	More Fragments (in IP)
MB	10^6 bytes
Mbps	10^6 bits per second
OAM	Operation, Administration and Maintenance (in SDH)
OSI	Open Systems Interconnection
PCF	Point Coordination Function (in 802.11 MAC)
PCM	Pulse Code Modulation
PSTN	Public Switched Telephone Network
RARP	Reverse Address Resolution Protocol (in IP)
SDH	Synchronous Digital Hierarchy
SIPP	Simple Internet Protocol Plus
SOH	Section Overhead (in SDH)
SONET	Synchronous Optical Network
STM	Synchronous Transport Module (in SDH)
STS	Synchronous Transport Signal (in SONET)
TCP	Transport Control Protocol
TDM	Time-Division Multiplexed

UDP	User Datagram Protocol
VC	Virtual Container (in SDH)
WWW	World Wide Web

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.fi/

University of Turku
• Department of Mathematical Sciences

Åbo Akademi University
• Department of Computer Science
• Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
• Institute of Information Systems Science