Communications protocol

From Wikipedia, the free encyclopedia

In telecommunications, a **communication protocol** is a system of rules that allow two or more entities of a communications system to transmit information via any kind of variation of a physical quantity. These are the rules or standard that defines the syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both.[1]

Communicating systems use well-defined formats (protocol) for exchanging various messages. Each message has an exact meaning intended to elicit a response from a range of possible responses pre-determined for that particular situation. The specified behavior is typically independent of how it is to be implemented. Communications protocols have to be agreed upon by the parties involved.[2] To reach agreement, a protocol may be developed into a technical standard. A programming language describes the same for computations, so there is a close analogy between protocols and programming languages: *protocols are to communications what programming languages are to computations*.[3]

Multiple protocols often describe different aspects of a single communication. A group of protocols designed to work together are known as a protocol suite; when implemented in software they are a protocol stack.

Most recent protocols are assigned by the IETF for Internet communications, and the IEEE, or the ISO organizations for other types. The ITU-T handles telecommunications protocols and formats for the PSTN. As the PSTN and Internet converge, the two sets of standards are also being driven towards convergence.

Contents

Communicating systems

The information exchanged between devices through a network, or other media is governed by rules and conventions that can be set out in technical specifications called communications protocol standards. The nature of a communication, the actual data exchanged and any state-dependent behaviors, is defined by these specifications.

In digital computing systems, the rules can be expressed by algorithms and data structures. Expressing the algorithms in a portable programming language makes the protocol software operating-system independent.

Operating systems usually contain a set of cooperating processes that manipulate shared data to communicate with each other. This communication is governed by well-understood protocols, which can be embedded in the process code itself.[4][5]

In contrast, because there is no common memory, communicating systems have to communicate with each other using a shared transmission medium. Transmission is not necessarily reliable, and individual systems may use different hardware or operating systems.

To implement a networking protocol, the protocol software modules are interfaced with a framework implemented on the machine's operating system. This framework implements the networking functionality of the operating system.[6] The best known frameworks are the TCP/IP model and the OSI model.

At the time the Internet was developed, layering had proven to be a successful design approach for both compiler and operating system design and, given the similarities between programming languages and communications protocols, layering was applied to the protocols as well.[7] This gave rise to the concept of layered protocols which nowadays forms the basis of protocol design.[8]

Systems typically do not use a single protocol to handle a transmission. Instead they use a set of cooperating protocols, sometimes called a protocol family or protocol suite.[9] Some of the best known protocol suites include: IPX/SPX, X.25, AX.25, AppleTalk and TCP/IP.

The protocols can be arranged based on functionality in groups, for instance there is a group of transport protocols. The functionalities are mapped onto the layers, each layer solving a distinct class of problems relating to, for instance: application-, transport-, internet- and network interface-functions.[10] To transmit a message, a protocol has to be selected from each layer, so some sort of multiplexing and demultiplexing takes place. The selection of the next protocol is accomplished by extending the message with a protocol selector for each layer.[11]

Basic requirements of protocols

Getting the data across a network is only part of the problem for a protocol. The data received has to be evaluated in the context of the progress of the conversation, so a protocol has to specify rules describing the context. These kind of rules are said to express the *syntax* of the communications. Other rules determine whether the data is meaningful for the context in which the exchange takes place. These kind of rules are said to express the *semantics* of the communications.

Messages are sent and received on communicating systems to establish communications. Protocols should therefore specify rules governing the transmission. In general, much of the following should be addressed:[12]

- *Data formats for data exchange*. Digital message bitstrings are exchanged. The bitstrings are divided in fields and each field carries information relevant to the protocol. Conceptually the bitstring is divided into two parts called the *header area* and the *data area*. The actual message is stored in the data area, so the header area contains the fields with more relevance to the protocol. Bitstrings longer than the maximum transmission unit (MTU) are divided in pieces of appropriate size.[13]
- *Address formats for data exchange*. Addresses are used to identify both the sender and the intended receiver(s). The addresses are stored in the header area of the bitstrings, allowing the receivers to determine whether the bitstrings are intended for themselves and should be processed or should be ignored. A connection between a sender and a receiver can be identified using an address pair *(sender address, receiver address)*. Usually some address values have special meanings. An all-*1*s address could be taken to mean an addressing of all stations on the network, so sending to this address would result in a broadcast on the local network. The rules describing the meanings of the address value are collectively called an *addressing scheme*.[14]
- *Address mapping*. Sometimes protocols need to map addresses of one scheme on addresses of another scheme. For instance to translate a logical IP address specified by the application to an Ethernet hardware address. This is referred to as *address mapping*.[15]
- *Routing*. When systems are not directly connected, intermediary systems along the *route* to the intended receiver(s) need to forward messages on behalf of the sender. On the Internet, the networks are connected using routers. This way of connecting networks is called *internetworking*.
- *Detection of transmission errors* is necessary on networks which cannot guarantee error-free operation. In a common approach, CRCs of the data area are added to the end of packets, making it possible for the receiver to detect differences caused by errors. The receiver rejects the packets on CRC differences and arranges somehow for retransmission.[16]
- *Acknowledgements* of correct reception of packets is required for connection-oriented communication. Acknowledgements are sent from receivers back to their respective senders.[17]

- *Loss of information - timeouts and retries*. Packets may be lost on the network or suffer from long delays. To cope with this, under some protocols, a sender may expect an acknowledgement of correct reception from the receiver within a certain amount of time. On timeouts, the sender must assume the packet was not received and retransmit it. In case of a permanently broken link, the retransmission has no effect so the number of retransmissions is limited. Exceeding the retry limit is considered an error.[18]
- *Direction of information flow* needs to be addressed if transmissions can only occur in one direction at a time as on half-duplex links. This is known as Media Access Control. Arrangements have to be made to accommodate the case when two parties want to gain control at the same time. [19]
- *Sequence control*. We have seen that long bitstrings are divided in pieces, and then sent on the network individually. The pieces may get lost or delayed or take different routes to their destination on some types of networks. As a result, pieces may arrive out of sequence. Retransmissions can result in duplicate pieces. By marking the pieces with sequence information at the sender, the receiver can determine what was lost or duplicated, ask for necessary retransmissions and reassemble the original message.[20]
- *Flow control* is needed when the sender transmits faster than the receiver or intermediate network equipment can process the transmissions. Flow control can be implemented by messaging from receiver to sender.[21]

Protocols and programming languages

Protocols are to communications what algorithms or programming languages are to computations.[3][22]

This analogy has important consequences for both the design and the development of protocols. One has to consider the fact that algorithms, programs and protocols are just different ways of describing expected behavior of interacting objects. A familiar example of a protocolling language is the HTML language used to describe web pages which are the actual web protocols.

In programming languages the association of *identifiers* to a *value* is termed a *definition*. Program text is structured using *block* constructs and definitions can be local to a block. The localized association of an identifier to a value established by a definition is termed a *binding* and the region of program text in which a binding is effective is known as its *scope*.[23] The computational state is kept using two components: the *environment*, used as a record of identifier bindings, and the *store*, which is used as a record of the effects of assignments.[24]

In communications, message values are transferred using transmission media. By analogy, the equivalent of a store would be a collection of transmission media, instead of a collection of memory locations. A valid assignment in a protocol (as an analog of programming language) could be *Ethernet:='message'* , meaning a message is to be broadcast on the local ethernet.

On a transmission medium there can be many receivers. For instance a mac-address identifies an ether network card on the transmission medium (the 'ether'). In our imaginary protocol, the assignment *ethernet[mac-address]:=message value* could therefore make sense.[25]

By extending the assignment statement of an existing programming language with the semantics

described, a protocolling language could easily be imagined.

Operating systems provide reliable communication and synchronization facilities for communicating objects confined to the same system by means of *system libraries*. A programmer using a general purpose programming language (like C or Ada) can use the routines in the libraries to implement a protocol, instead of using a dedicated protocolling language.

Universal protocols

> The nice thing about standards is that you have so many to choose from.
>
> —Andrew S. Tanenbaum in *Computer Networks*[26]

Despite their numbers, networking protocols show little variety, because all networking protocols use the same underlying principles and concepts, in the same way. So, the use of a general purpose programming language would yield a large number of applications only differing in the details.[27] A suitably defined (dedicated) protocolling language would therefore have little syntax, perhaps just enough to specify some parameters or optional modes of operation, because its virtual machine would have incorporated all possible principles and concepts making the virtual machine itself a *universal* protocol. The protocolling language would have some syntax and a lot of semantics describing this universal protocol and would therefore in effect be a protocol, hardly differing from this universal networking protocol. In this (networking) context a protocol is a language.

The notion of a universal networking protocol provides a rationale for standardization of networking protocols; assuming the existence of a universal networking protocol, development of protocol standards using a consensus model (the agreement of a group of experts) might be a viable way to coordinate protocol design efforts.

Networking protocols operate in very heterogeneous environments consisting of very different network technologies and a (possibly) very rich set of applications, so a single universal protocol would be very hard to design and implement correctly. Instead, the IETF decided to reduce complexity by assuming a relatively simple network architecture allowing decomposition of the single universal networking protocol into two generic protocols, TCP and IP, and two classes of specific protocols, one dealing with the low-level network details and one dealing with the high-level details of common network applications (remote login, file transfer, email and web browsing). ISO choose a similar but more general path, allowing other network architectures, to standardize protocols.

Protocol design

Systems engineering principles have been applied to create a set of common network protocol design principles.

Communicating systems operate in parallel. The programming tools and techniques for dealing with parallel processes are collectively called *concurrent programming*. Concurrent programming only deals with the synchronization of communication. The syntax and semantics of the communication governed by a low-level protocol usually have modest complexity, so they can be coded with relative ease. High-level protocols with relatively large complexity could however merit the implementation of language interpreters. An example of the latter case is the HTML language.

Concurrent programming has traditionally been a topic in operating systems theory texts.[28] Formal verification seems indispensable, because concurrent programs are notorious for the hidden and sophisticated bugs they contain.[29] A mathematical approach to the study of concurrency and communication is referred to as *Communicating Sequential Processes* (CSP).[30] Concurrency can also be modelled using finite state machines like Mealy and Moore machines. Mealy and Moore machines are in use as design tools in digital electronics systems, which we encounter in the form of hardware used in telecommunications or electronic devices in general.[31]

This kind of design can be a bit of a challenge to say the least, so it is important to keep things simple. For the Internet protocols, in particular and in retrospect, this meant a basis for protocol design was needed to allow decomposition of protocols into much simpler, cooperating protocols.

A *basis* for protocol design

Systems do not use a single protocol to handle a transmission. Instead they use a set of cooperating protocols, sometimes called a protocol family or protocol suite.[9] To cooperate the protocols have to communicate with each other, so some kind of conceptual framework is needed to make this communication possible. Also note that software is needed to implement both the 'xfer-mechanism' and a protocol (no protocol, no communication).

In literature there are numerous references to the analogies between computer communication and programming. By analogy we could say that the aforementioned 'xfer-mechanism' is comparable to a *cpu*; a 'xfer-mechanism' performs communications and a *cpu* performs computations and the 'framework' introduces something that allows the protocols to be designed independent of one another by providing separate execution environments for them. Furthermore, it is repeatedly stated that *protocols are to computer communication what programming languages are to computation*.[32][33]

Layering

In modern protocol design, protocols are "layered". Layering is a design principle which divides the protocol design into a number of smaller parts, each of which accomplishes a particular sub-task, and interacts with the other parts of the protocol only in a small number of well-defined ways.

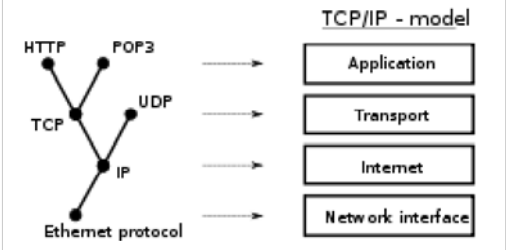

Figure 2. The TCP/IP model or Internet layering scheme and its relation to some common protocols.

Layering allows the parts of a protocol to be designed and tested without a combinatorial explosion of cases, keeping each design relatively simple. Layering also permits familiar protocols to be adapted to unusual circumstances. For example, the mail protocol above can be adapted to send messages to aircraft. Just change the V.42 modem protocol to the INMARS LAPD data protocol used by the international marine radio satellites.

The communications protocols in use on the Internet are designed to function in very diverse and complex settings. To ease design, communications protocols are structured using a layering scheme as a basis. Instead of using a single universal protocol to handle all transmission tasks, a set of cooperating protocols fitting the layering scheme is used.[34] The layering scheme in use on the Internet is called the TCP/IP model. The actual protocols are collectively called the Internet protocol suite. The Internet Engineering Task Force (*IETF*) is responsible for this design.

Another reference model used for layering is the OSI seven layer model, which can be applied to any protocol, not just the OSI protocols. In particular, the Internet Protocol can be analysed using the OSI model.

Typically, a hardware delivery mechanism layer is used to build a connectionless packet delivery system on top of which a reliable transport layer is built, on top of which is the application software. Layers below and above these can be defined, and protocols are very often stacked to give *tunnelling*, for example the internet protocol can be tunnelled across an ATM network protocol to provide connectivity by layering the internet protocol on top of the ATM protocol transport layer.

The number of layers of a layering scheme and the way the layers are defined can have a drastic impact on the protocols involved. This is where the analogies come into play for the TCP/IP model, because the designers of TCP/IP employed the same techniques used to conquer the complexity of programming language *compilers* (design by analogy) in the *implementation* of its protocols and its layering scheme.[35]

For example, one layer might describe how to encode text (with ASCII, say), while another describes how to inquire for messages (with the Internet's simple mail transfer protocol, for example), while another may detect and retry errors (with the Internet's transmission control protocol), another handles addressing (say with IP, the Internet Protocol), another handles the encapsulation of that data into a stream of bits (for example, with the point-to-point protocol), and another handles the electrical encoding of the bits, (with a V.42 modem, for example).

Protocol layering

Protocol layering now forms the basis of protocol design.[8] It allows the decomposition of single, complex protocols into simpler, cooperating protocols, but it is also a functional decomposition, because each protocol belongs to a functional class, called a *protocol layer*.[34] The protocol layers each solve a distinct class of communication problems. The Internet protocol suite consists of the following layers: application-, transport-, internet- and network interface-functions.[10] Together, the layers make up a *layering scheme* or *model*.

In computations, we have algorithms and data, and in communications, we have protocols and messages, so the analog of a data flow diagram would be some kind of message flow diagram.[22] To visualize protocol layering and protocol suites, a diagram of the message flows in and between two systems, A and B, is shown in figure 3.

The systems both make use of the same protocol suite. The vertical flows (and protocols) are *in system* and the horizontal message flows (and protocols) are *between* systems. The message flows are governed by rules, and data formats specified by protocols. The blue lines therefore mark the boundaries of the (horizontal) protocol layers.

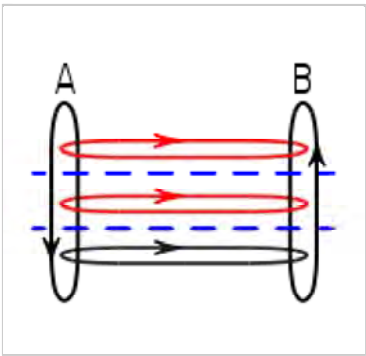

Figure 3. Message flows using a protocol suite. Black loops show the actual messaging loops, red loops are the effective communications between layers enabled by the lower layers.

The vertical protocols are not layered because they don't obey the *protocol layering principle* which states that *a layered protocol is designed so that layer* n *at the destination receives exactly the same object sent by layer* n *at the source*. The horizontal protocols are *layered protocols* and all belong to the protocol suite. Layered protocols allow the protocol designer to concentrate on one layer at a time, without worrying about how other layers perform.[33]

The vertical protocols need not be the same protocols on both systems, but they have to satisfy some minimal assumptions to ensure the protocol layering principle holds for the layered protocols. This can be achieved using a technique called *Encapsulation*.[36]

Usually, a message or a stream of data is divided into small pieces, called *messages* or *streams*, *packets*, *IP datagrams* or *network frames* depending on the layer in which the pieces are to be transmitted. The pieces contain a *header area* and a *data area*. The data in the header area identifies the source and the destination on the network of the packet, the protocol, and other data meaningful to the protocol like CRC's of the data to be sent, data length, and a timestamp.[37][38]

The rule enforced by the vertical protocols is that the pieces for transmission are to be *encapsulated* in the data area of all lower protocols on the sending side and the reverse is to happen on the receiving side. The result is that at the lowest level the piece looks like this: 'Header1,Header2,Header3,data' and in the layer directly above it: 'Header2,Header3,data' and in the top layer: 'Header3,data', both on the sending and receiving side. This rule therefore ensures that the protocol layering principle holds and effectively virtualizes all but the lowest transmission lines, so for this reason some message flows are coloured red in figure 3.

To ensure both sides use the same protocol, the pieces also carry data identifying the protocol in their header.

The design of the protocol layering and the network (or Internet) architecture are interrelated, so one cannot be designed without the other.[39] Some of the more important features in this respect of the Internet architecture and the network services it provides are described next.

- The Internet offers *universal interconnection*, which means that any pair of computers connected to the Internet is allowed to communicate. Each computer is identified by an *address* on the Internet. All the interconnected physical networks appear to the user as a single large network. This interconnection scheme is called an *internetwork* or *internet*.[40]
- Conceptually, an *Internet addresses* consists of a *netid* and a *hostid*. The netid identifies a network and the hostid identifies a host. The term host is misleading in that an individual computer can have multiple network interfaces each having its own Internet address. An Internet Address

identifies a connection to the network, not an individual computer.[41] The netid is used by routers to decide where to send a packet.[42]

- *Network technology independence* is achieved using the low-level *address resolution protocol* (ARP) which is used to map Internet addresses to physical addresses. The mapping is called *address resolution*. This way physical addresses are only used by the protocols of the network interface layer.[43] The TCP/IP protocols can make use of almost any underlying communication technology.[44]
- *Physical networks are interconnected by routers*. Routers forward packets between interconnected networks making it possible for hosts to reach hosts on other physical networks. The message flows between two communicating system A and B in the presence of a router R are illustrated in figure 4. Datagrams are passed from router to router until a router is reached that can deliver the datagram on a physically attached network (called *direct delivery*).[45] To decide whether a datagram is to be delivered directly or is to be sent to a router closer to the destination, a table called the *IP routing table* is consulted. The table consists of pairs of networkids and the paths to be taken to reach known networks. The path can be an indication that the datagram should be delivered directly or it can be the address of a router known to be closer to the destination.[46] A special entry can specify that a default router is chosen when there are no known paths.[47]

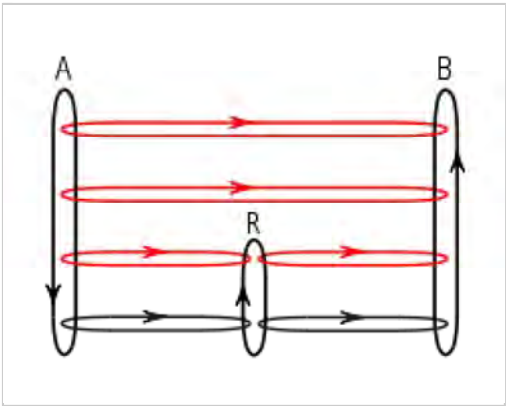

Figure 4. Message flows in the presence of a router

- *All networks are treated equal*. A LAN, a WAN or a point-to-point link between two computers are all considered as one network.[48]
- A *Connectionless packet delivery (or packet-switched) system (or service)* is offered by the Internet, because it adapts well to different hardware, including best-effort delivery mechanisms like the *ethernet*. Connectionless delivery means that the messages or streams are divided into pieces that are *multiplexed* separately on the high speed intermachine connections allowing the connections to be used concurrently. Each piece carries information identifying the destination. The delivery of packets is said to be *unreliable*, because packets may be lost, duplicated, delayed or delivered out of order without notice to the sender or receiver. Unreliability arises only when resources are exhausted or underlying networks fail.[49] The unreliable connectionless delivery system is defined by the *Internet Protocol* (IP). The protocol also specifies the *routing* function, which chooses a path over which data will be sent.[50] It is also possible to use TCP/IP protocols on *connection oriented systems*. Connection oriented systems build up *virtual circuits* (paths for exclusive use) between senders and receivers. Once built up the IP datagrams are sent as if they were data through the virtual circuits and forwarded (as data) to the IP protocol modules. This technique, called *tunneling*, can be used on X.25 networks and ATM networks.[51]
- A *reliable stream transport service* using the unreliable connectionless packet delivery service is defined by the *transmission control protocol* (TCP). The services are layered as well and the application programs residing in the layer above it, called the *application services*, can make use of TCP.[52] Programs wishing to interact with the packet delivery system itself can do so using the *user datagram protocol* (UDP).[53]

Software layering

Having established the protocol layering and the protocols, the protocol designer can now resume with the software design. The software has a layered organization and its relationship with protocol layering is visualized in figure 5.

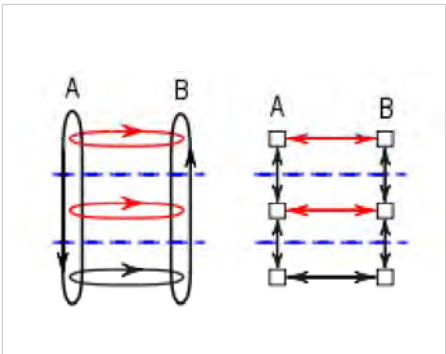

Figure 5: Protocol and software layering

The software modules implementing the protocols are represented by cubes. The information flow between the modules is represented by arrows. The (top two horizontal) red arrows are virtual. The blue lines mark the layer boundaries.

To send a message on system A, the top module interacts with the module directly below it and hands over the message to be encapsulated. This module reacts by encapsulating the message in its own data area and filling in its header data in accordance with the protocol it implements and interacts with the module below it by handing over this newly formed message whenever appropriate. The bottom module directly interacts with the bottom module of system B, so the message is sent across. On the receiving system B the reverse happens, so ultimately (and assuming there were no transmission errors or protocol violations etc.) the message gets delivered in its original form to the topmodule of system B.[54]
On protocol errors, a receiving module discards the piece it has received and reports back the error condition to the original source of the piece on the same layer by handing the error message down or in case of the bottom module sending it across.[55]
The division of the message or stream of data into pieces and the subsequent reassembly are handled in the layer that introduced the division/reassembly. The reassembly is done at the destination (i.e. not on any intermediate routers).[56]

TCP/IP software is organized in four layers.[57]

- *Application layer*. At the highest layer, the services available across a TCP/IP internet are accessed by application programs. The application chooses the style of transport to be used which can be a sequence of individual messages or a continuous stream of bytes. The application program passes data to the *transport layer* for delivery.
- *Transport layer*. The transport layer provides communication from one application to another. The transport layer may regulate flow of information and provide reliable transport, ensuring that data arrives without error and in sequence. To do so, the receiving side sends back acknowledgments and the sending side retransmits lost pieces called packets. The stream of data is divided into packets by the module and each packet is passed along with a destination address to the next layer for transmission. The layer must accept data from many applications concurrently and therefore also includes codes in the packet header to identify the sending and receiving application program.
- *Internet layer*. The Internet layer handles the communication between machines. Packets to be sent are accepted from the transport layer along with an identification of the receiving machine. The packets are encapsulated in IP datagrams and the datagram headers are filled. A routing algorithm is used to determine if the datagram should be delivered directly or sent to a router. The datagram is passed to the appropriate network interface for transmission. Incoming datagrams are

checked for validity and the routing algorithm is used to decide whether the datagram should be processed locally or forwarded. If the datagram is addressed to the local machine, the datagram header is deleted and the appropriate transport protocol for the packet is chosen. ICMP error and control messages are handled as well in this layer.

- *Network interface layer*. The network interface layer is responsible for accepting IP datagrams and transmitting them over a specific network. A network interface may consist of a device driver or a complex subsystem that uses its own data link protocol.

Program translation has been divided into four subproblems: compiler, assembler, link editor, and loader. As a result, the translation software is layered as well, allowing the software layers to be designed independently. Noting that the ways to conquer the complexity of program translation could readily be applied to protocols because of the analogy between programming languages and protocols, the designers of the TCP/IP protocol suite were keen on imposing the same layering on the software framework. This can be seen in the TCP/IP layering by considering the translation of a *pascal program* (message) that is compiled (function of the application layer) into an *assembler program* that is assembled (function of the transport layer) to *object code* (pieces) that is linked (function of the Internet layer) together with *library object code* (routing table) by the link editor, producing *relocatable machine code* (datagram) that is passed to the loader which fills in the memory locations (ethernet addresses) to produce *executable code* (network frame) to be loaded (function of the network interface layer) into physical memory (transmission medium). To show just how closely the analogy fits, the terms between parentheses in the previous sentence denote the relevant analogs and the terms written *cursively* denote data representations. Program translation forms a linear sequence, because each layer's output is passed as input to the next layer. Furthermore, the translation process involves multiple data representations. We see the same thing happening in protocol software where multiple protocols define the data representations of the data passed between the software modules.[35]

The network interface layer uses physical addresses and all the other layers only use IP addresses. The boundary between network interface layer and Internet layer is called the *high-level protocol address boundary*.[58] The modules below the application layer are generally considered part of the operating system. Passing data between these modules is much less expensive than passing data between an application program and the transport layer. The boundary between application layer and transport layer is called the *operating system boundary*.[59]

Strict layering

Strictly adhering to a layered model, a practice known as strict layering, is not always the best approach to networking.[60] Strict layering, can have a serious impact on the performance of the implementation, so there is at least a trade-off between simplicity and performance.[61] Another, perhaps more important point can be shown by considering the fact that some of the protocols in the Internet Protocol Suite cannot be expressed using the TCP/IP model, in other words some of the protocols behave in ways not described by the model.[62] To improve on the model, an offending protocol could, perhaps be split up into two protocols, at the cost of one or two extra layers, but there is a hidden caveat, because the model is also used to provide a conceptual view on the suite for the intended users. There is a trade-off to be made here between preciseness for the designer and clarity for the intended user.[63]

Formal specification

Formal ways for describing the syntax of the communications are Abstract Syntax Notation One (an ISO standard) or Augmented Backus-Naur form (an IETF standard).

Finite state machine models[64][65] and communicating finite-state machines[66] are used to formally describe the possible interactions of the protocol.

Protocol development

For communication to take place, protocols have to be agreed upon. Recall that in digital computing systems, the rules can be expressed by algorithms and datastructures, raising the opportunity for hardware independence. Expressing the algorithms in a portable programming language, makes the protocol software operating system independent. The source code could be considered a protocol specification. This form of specification, however is not suitable for the parties involved.

For one thing, this would enforce a source on all parties and for another, proprietary software producers would not accept this. By describing the software interfaces of the modules on paper and agreeing on the interfaces, implementers are free to do it their way. This is referred to as source independence. By specifying the algorithms on paper and detailing hardware dependencies in an unambiguous way, a *paper draft* is created, that when adhered to and published, ensures interoperability between software and hardware.

Such a paper draft can be developed into a *protocol standard* by getting the approval of a *standards organization*. To get the approval the paper draft needs to enter and successfully complete the *standardization process*. This activity is referred to as *protocol development*. The members of the standards organization agree to adhere to the standard on a voluntary basis. Often the members are in control of large market-shares relevant to the protocol and in many cases, standards are enforced by law or the government, because they are thought to serve an important public interest, so getting approval can be very important for the protocol.

It should be noted though that in some cases protocol standards are not sufficient to gain widespread acceptance i.e. sometimes the source code needs to be disclosed and enforced by law or the government in the interest of the public.

The need for protocol standards

The need for protocol standards can be shown by looking at what happened to the bi-sync protocol (BSC) invented by IBM. BSC is an early link-level protocol used to connect two separate nodes. It was originally not intended to be used in a multinode network, but doing so revealed several deficiencies of the protocol. In the absence of standardization, manufacturers and organizations felt free to 'enhance' the protocol, creating incompatible versions on their networks. In some cases, this was deliberately done to discourage users from using equipment from other manufacturers. There are more than 50 variants of the original bi-sync protocol. One can assume, that a standard would have prevented at least some of this from happening.[6]

In some cases, protocols gain market dominance without going through a standardization process. Such protocols are referred to as *de facto standards*. De facto standards are common in emerging markets, niche markets, or markets that are monopolized (or oligopolized). They can hold a market in a very negative grip, especially when used to scare away competition. From a historical perspective, standardization should be seen as a measure to counteract the ill-effects of de facto standards. Positive exceptions exist; a 'de facto standard' operating system like GNU/Linux does not have this negative grip on its market, because the sources are published and maintained in an open way, thus inviting competition. Standardization is therefore not the only solution for *open systems interconnection*.

Standards organizations

Some of the standards organizations of relevance for communications protocols are the International Organization for Standardization (ISO), the International Telecommunication Union (ITU), the Institute of Electrical and Electronics Engineers (IEEE), and the Internet Engineering Task Force (IETF). The IETF maintains the protocols in use on the Internet. The IEEE controls many software and hardware protocols in the electronics industry for commercial and consumer devices. The ITU is an umbrella organization of telecommunication engineers designing the public switched telephone network (PSTN), as well as many radio communications systems. For marine electronics the NMEA standards are used. The World Wide Web Consortium (W3C) produces protocols and standards for Web technologies.

International standards organizations are supposed to be more impartial than local organizations with a national or commercial self-interest to consider. Standards organizations also do research and development for standards of the future. In practice, the standards organizations mentioned, cooperate closely with each other.[67]

The standardization process

The standardization process starts off with ISO commissioning a sub-committee workgroup. The workgroup issues working drafts and discussion documents to interested parties (including other standards bodies) in order to provoke discussion and comments. This will generate a lot of questions, much discussion and usually some disagreement on what the standard should provide and if it can satisfy all needs (usually not). All conflicting views should be taken into account, often by way of compromise, to progress to a *draft proposal* of the working group.

The draft proposal is discussed by the member countries' standard bodies and other organizations within each country. Comments and suggestions are collated and national views will be formulated, before the members of ISO vote on the proposal. If rejected, the draft proposal has to consider the objections and counter-proposals to create a new draft proposal for another vote. After a lot of feedback, modification, and compromise the proposal reaches the status of a *draft international standard*, and ultimately an *international standard*.

The process normally takes several years to complete. The original paper draft created by the designer will differ substantially from the standard, and will contain some of the following 'features':

- Various optional modes of operation, for example to allow for setup of different packet sizes at startup time, because the parties could not reach consensus on the optimum packet size.

- Parameters that are left undefined or allowed to take on values of a defined set at the discretion of the implementor. This often reflects conflicting views of some of the members.
- Parameters reserved for future use, reflecting that the members agreed the facility should be provided, but could not reach agreement on how this should be done in the available time.
- Various inconsistencies and ambiguities will inevitably be found when implementing the standard.

International standards are reissued periodically to handle the deficiencies and reflect changing views on the subject.[68]

Future of standardization (OSI)

A lesson learned from ARPANET (the predecessor of the Internet) is that standardization of protocols is not enough, because protocols also need a framework to operate. It is therefore important to develop a general-purpose, future-proof framework suitable for *structured protocols* (such as layered protocols) and their standardization. This would prevent protocol standards with overlapping functionality and would allow clear definition of the responsibilities of a protocol at the different levels (layers).[69] This gave rise to the OSI *Open Systems Interconnection reference model* (RM/OSI), which is used as a framework for the design of standard protocols and services conforming to the various layer specifications.[70]

In the OSI model, communicating systems are assumed to be connected by an underlying physical medium providing a basic (and unspecified) transmission mechanism. The layers above it are numbered (from one to seven); the n^{th} layer is referred to as (n)-layer. Each layer provides service to the layer above it (or at the top to the application process) using the services of the layer immediately below it. The layers communicate with each other by means of an interface, called a *service access point*. Corresponding layers at each system are called *peer entities*. To communicate, two peer entities at a given layer use an (n)-protocol, which is implemented by using services of the (n-1)-layer. When systems are not directly connected, intermediate peer entities (called *relays*) are used. An *address* uniquely identifies a service access point. The address naming domains need not be restricted to one layer, so it is possible to use just one naming domain for all layers.[71] For each layer there are two types of standards: protocol standards defining how peer entities at a given layer communicate, and service standards defining how a given layer communicates with the layer above it.

In the original version of RM/OSI, the layers and their functionality are (from highest to lowest layer):

- The *application layer* may provide the following services to the application processes: identification of the intended communication partners, establishment of the necessary authority to communicate, determination of availability and authentication of the partners, agreement on privacy mechanisms for the communication, agreement on responsibility for error recovery and procedures for ensuring data integrity, synchronization between cooperating application processes, identification of any constraints on syntax (e.g. character sets and data structures), determination of cost and acceptable quality of service, selection of the dialogue discipline, including required logon and logoff procedures.[72]
- The *presentation layer* may provide the following services to the application layer: a request for the establishment of a session, data transfer, negotiation of the syntax to be used between the

application layers, any necessary syntax transformations, formatting and special purpose transformations (e.g. data compression and data encryption).[73]

- The *session layer* may provide the following services to the presentation layer: establishment and release of session connections, normal and expedited data exchange, a quarantine service which allows the sending presentation entity to instruct the receiving session entity not to release data to its presentation entity without permission, interaction management so presentation entities can control whose turn it is to perform certain control functions, resynchronization of a session connection, reporting of unrecoverable exceptions to the presentation entity.[74]
- The *transport layer* provides reliable and transparent data transfer in a cost-effective way as required by the selected quality of service. It may support the multiplexing of several transport connections on to one network connection or split one transport connection into several network connections.[75]
- The *network layer* does the setup, maintenance and release of network paths between transport peer entities. When relays are needed, routing and relay functions are provided by this layer. The quality of service is negotiated between network and transport entities at the time the connection is set up. This layer is also responsible for network congestion control.[76]
- The *data link layer* does the setup, maintenance and release of data link connections. Errors occurring in the physical layer are detected and may be corrected. Errors are reported to the network layer. The exchange of data link units (including flow control) is defined by this layer.[77]
- The *physical layer* describes details like the electrical characteristics of the physical connection, the transmission techniques used, and the setup, maintenance and clearing of physical connections.[78]

In contrast to the TCP/IP layering scheme, which assumes a connectionless network, RM/OSI assumed a connection-oriented network. Connection-oriented networks are more suitable for wide area networks and connectionless networks are more suitable for local area networks. Using connections to communicate implies some form of session and (virtual) circuits, hence the (in the TCP/IP model lacking) session layer. The constituent members of ISO were mostly concerned with wide area networks, so development of RM/OSI concentrated on connection oriented networks and connectionless networks were only mentioned in an addendum to RM/OSI.[79] At the time, the IETF had to cope with this and the fact that the Internet needed protocols which simply were not there. As a result, the IETF developed its own standardization process based on "rough consensus and running code".[80]

The standardization process is described by RFC2026 (http://tools.ietf.org/html/rfc2026).

Nowadays, the IETF has become a standards organization for the protocols in use on the Internet. RM/OSI has extended its model to include connectionless services and because of this, both TCP and IP could be developed into international standards.

Taxonomies

Classification schemes for protocols usually focus on domain of use and function. As an example of domain of use, connection-oriented protocols and connectionless protocols are used on connection-oriented networks and connectionless networks respectively. For an example of function consider a tunneling protocol, which is used to encapsulate packets in a high-level protocol, so the packets can be passed across a transport system using the high-level protocol.

A *layering scheme* combines both function and domain of use. The dominant layering schemes are the ones proposed by the IETF and by ISO. Despite the fact that the underlying assumptions of the layering schemes are different enough to warrant distinguishing the two, it is a common practice to compare the two by relating common protocols to the layers of the two schemes.[81] For an example of this practice see: List of network protocols.

The layering scheme from the IETF is called *Internet layering* or *TCP/IP layering*. The functionality of the layers has been described in the section on software layering and an overview of protocols using this scheme is given in the article on Internet protocols.

The layering scheme from ISO is called *the OSI model* or *ISO layering*. The functionality of the layers has been described in the section on the future of standardization and an overview of protocols using this scheme is given in the article on OSI protocols.

Examples of protocols

Protocol stacks or families include multiple interacting protocols:

- PARC Universal Packet
- Internet protocol suite
- AppleTalk
- DECnet
- IPX/SPX
- Open Systems Interconnection (OSI)
- Systems Network Architecture (SNA)

The Internet Protocol is used in concert with other protocols within the Internet protocol suite, notable components of which include:

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Internet Control Message Protocol (ICMP)
- Hypertext Transfer Protocol (HTTP)
- Post Office Protocol (POP)
- File Transfer Protocol (FTP)
- Internet Message Access Protocol (IMAP)

Other instances of high level interaction protocols are:

- General Inter-ORB Protocol (GIOP)
- Java remote method invocation (RMI)
- Distributed Component Object Model (DCOM)
- Dynamic Data Exchange (DDE)
- SOAP

See also

- Application programming interface

Notes

1. Licesio J. Rodríguez-Aragón: *Tema 4: Internet y Teleinformática (http://www.uclm.es/profesorado/licesio/Docencia/IB/IBTema4.pdf)*. retrieved 2013-04-24. **(Spanish)**
2. *Protocol*, Encyclopedia Britannica, retrieved 2012-09-24
3. Comer 2000, Sect. 11.2 - The Need For Multiple Protocols, p. 177, "They (protocols) are to communication what programming languages are to computation"
4. Ben-Ari 1982, chapter 2 - The concurrent programming abstraction, p. 18-19, states the same.
5. Ben-Ari 1982, Section 2.7 - Summary, p. 27, summarizes the concurrent programming abstraction.
6. Marsden 1986, Section 6.1 - Why are standards necessary?, p. 64-65, uses BSC as an example to show the need for both standard protocols and a standard framework.
7. Comer 2000, Sect. 11.2 - The Need For Multiple Protocols, p. 177, explains this by drawing analogies between computer communication and programming languages.
8. Sect. 11.10 - The Disadvantage Of Layering, p. 192, states: layering forms the basis for protocol design.
9. Comer 2000, Sect. 11.2 - The Need For Multiple Protocols, p. 177, states the same.
10. Comer 2000, Sect. 11.3 - The Conceptual Layers Of Protocol Software, p. 178, "Each layer takes responsibility for handling one part of the problem."
11. Comer 2000, Sect. 11.11 - The Basic Idea Behind Multiplexing And Demultiplexing, p. 192, states the same.
12. Marsden 1986, Chapter 3 - Fundamental protocol concepts and problem areas, p. 26-42, explains much of the following.
13. Comer 2000, Sect. 7.7.4 - Datagram Size, Network MTU, and Fragmentation, p. 104, Explains fragmentation and the effect on the header of the fragments.
14. Comer 2000, Chapter 4 - Classful Internet Addresses, p. 64-67;71.
15. Marsden 1986, Section 14.3 - Layering concepts and general definitions, p. 187, explains address mapping.
16. Marsden 1986, Section 3.2 - Detection and transmission errors, p. 27, explains the advantages of backward error correction.
17. Marsden 1986, Section 3.3 - Acknowledgement, p. 28-33, explains the advantages of positive only acknowledgement and mentions datagram protocols as exceptions.
18. Marsden 1986, Section 3.4 - Loss of information - timeouts and retries, p. 33-34.
19. Marsden 1986, Section 3.5 - Direction of information flow, p. 34-35, explains master/slave and the negotiations to gain control.
20. Marsden 1986, Section 3.6 - Sequence control, p. 35-36, explains how packets get lost and how sequencing solves this.
21. Marsden 1986, Section 3.7 - Flow control, p. 36-38.
22. Comer 2000, Sect. 1.3 - Internet Services, p. 3, "Protocols are to communication what algorithms are to computation"
23. Tennent 1981, Section 2.3.1 - Definitions, p.15, defines scope and binding.
24. Tennent 1981, Section 2.3.2 Environments and stores, p.16, the semantics of blocks and definitions are described using environments and stores.
25. Hoare (1985), Ch. 4 - Communication, p. 133, In the introduction: a communication is an event described by a pair c.v where c is the name of the communication channel and v is the value of the message.
26. Tanenbaum, Andrew S. (2003). *Computer networks*. Prentice Hall Professional. p. 235. ISBN 978-0-13-066102-9. Retrieved 22 June 2011.
27. Comer 2000, Foreword To The First Edition By The Late Jon Postel, xxv, "The principles of architecture, layering, multiplexing, encapsulation, addressing and address mapping, routing, and naming are quite similar in any protocol suite, though of course, different in detail.".
28. Ben-Ari 1982, in his preface, p. xiii.
29. Ben-Ari 1982, in his preface, p. xiv.

30. Hoare 1985, Chapter 4 - Communication, p. 133, deals with communication.
31. S. Srinivasan, NPTEL courses:::: Electronics & Communication Engineering :: Digital Circuits and Systems, available online: http://nptel.iitm.ac.in/video.php?courseId=1005&p=3
32. Comer 2000, Sect. 11.2 - The Need For Multiple Protocols, p. 177, states more or less the same, using other analogies.
33. Comer 2000, Sect. 11.7 - The Protocol Layering Principle, p. 187, explains layered protocols.
34. Comer 2000, Sect. 11.2 - The Need For Multiple Protocols, p. 177, introduces the decomposition in layers.
35. Comer 2000, Sect. 11.2 - The need for multiple protocols, p. 178, explains similarities protocol software and compiler, assembler, linker, loader.
36. Comer 2000, Glossary of Internetworking terms, p.686: term encapsulation.
37. Comer 2000, Sect. 11.5.1 - The TCP/IP 5-Layer Reference Model, p. 184, Describes the transformations of messages or streams that can be observed in the protocol layers.
38. Comer 2000, Sect. 2.4.10 - Ethernet Frame Format, p. 30, Ethernet frames are used as an example for administrative data for the protocol itself.
39. Comer 2000, Sect. 11.4 - Functionality Of The Layers, p. 181, states the same about the software organization.
40. Comer 2000, Sect. 3.3 - Network-Level Interconnection, p. 55, explains universal interconnection and internetworking.
41. Comer 2000, Sect. 4.4 - Addresses Specify Network Connections, p. 86, explains this.
42. Comer 2000, Sect. 4.3 - The Original Classful Addressing Scheme, p. 64, explains the address scheme, netid and routing.
43. Comer 2000, Sect. 5.13 - Summary, p. 86, explains ARP.
44. Comer 2000, Sect. 2.11 - Other Technologies Over Which TCP/IP Has Been Used, p. 46, states the same.
45. Comer 2000, Sect. 8.3.2 - Indirect Delivery, p. 118, states the same.
46. Comer 2000, Sect. 8.5 - Next-Hop Routing, p. 120, gives details on the routing table.
47. Comer 2000, Sect. 8.6 - Default Routes, p. 121, explains default routing and its use.
48. Comer 2000, Sect. 3.8 - All Networks Are Equal, p. 59, states the same.
49. Comer 2000, Sect. 7.5 - Connectionless Delivery System, p. 97, explains the delivery system.
50. Comer 2000, Sect. 7.6 - Purposes Of The Internet Protocol, p. 97, states the same.
51. Comer 2000, Sect. 2.11.1 - X25NET And Tunnels, p. 46-47, explains tunneling X.25 and mentions ATM.
52. Comer 2000, Sect. 13.1 - Introduction, p. 209, introduces TCP.
53. Comer 2000, Sect. 12.10 - Summary, p. 206, explains UDP.
54. Comer 2000, Sect. 11.3 - The Conceptual Layers Of Protocol Software, p. 179, the first two paragraphs describe the sending of a message through successive layers.
55. Comer 2000, Sect. 9.3 - Error Reporting vs. Error Correction, p. 131, describes the ICMP protocol that is used to handle datagram errors.
56. Comer 2000, Sect. 7.7.5 - Reassembly Of Fragments, p. 104, describes reassembly of datagrams.
57. Comer 2000, Sect. 11.5.1 - The TCP/IP 5-Layer Reference Model, p. 184, explains functionality of the layers.
58. Comer 2000, Sect. 11.9.1 - High-Level Protocol Boundary, p. 191, describes the boundary.
59. Comer 2000, Sect. 11.9.1 - Operating System Boundary, p. 192, describes the operating system boundary.
60. IETF 1989, Sect 1.3.1 - Organization, p. 15, 2nd paragraph: many design choices involve creative "breaking" of strict layering.
61. Comer 2000, Sect. 11.10 - The Disadvantage Of Layering, p. 192, explains why "strict layering can be extremely inefficient" giving examples of optimizations.
62. IETF 1989, Sect 1.3.1 - Organization, p. 15, 2nd paragraph, explaining why "strict layering is an imperfect model"
63. IETF 1989, Sect 1.3.1 - Organization, p. 15, states: This layerist organization was chosen for simplicity and clarity.
64. Bochmann, G. (1978). "Finite state description of communication protocols". *Computer Networks (1976)*. **2** (4–5): 361–201. doi:10.1016/0376-5075(78)90015-6.
65. Comer 2000, Glossary of Internetworking Terms and Abbreviations, p. 704, term protocol.
66. Brand, Daniel; Zafiropulo, Pitro (1983). "On Communicating Finite-State Machines". *Journal of the ACM*. **30** (2): 323. doi:10.1145/322374.322380.
67. Marsden 1986, Section 6.3 - Advantages of standardisation, p. 66-67, states the same.

68. Marsden 1986, Section 6.4 - Some problems with standardisation, p. 67, follows HDLC to illustrate the process.
69. Marsden 1986, Section 6.1 - Why are standards necessary?, p. 65, explains lessons learned from ARPANET.
70. Marsden 1986, Section 14.1 - Introduction, p. 181, introduces OSI.
71. Marsden 1986, Section 14.3 - Layering concepts and general definitions, p. 183-185, explains terminology.
72. Marsden 1986, Section 14.4 - The application layer, p. 188, explains this.
73. Marsden 1986, Section 14.5 - The presentation layer, p. 189, explains this.
74. Marsden 1986, Section 14.6 - The session layer, p. 190, explains this.
75. Marsden 1986, Section 14.7 - The transport layer, p. 191, explains this.
76. Marsden 1986, Section 14.8 - The network layer, p. 192, explains this.
77. Marsden 1986, Section 14.9 - The data link layer, p. 194, explains this.
78. Marsden 1986, Section 14.10 - The physical layer, p. 195, explains this.
79. Marsden 1986, Section 14.11 - Connectionless mode and RM/OSI, p. 195, mentions this.
80. Comer 2000, Section 1.9 - Internet Protocols And Standardization, p. 12, explains why the IETF did not use existing protocols.
81. Comer 2000, Sect. 11.5.1 - The TCP/IP 5-Layer Reference Model, p. 183, states the same.

References

- Radia Perlman: *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*. 2nd Edition. Addison-Wesley 1999, ISBN 0-201-63448-1. In particular Ch. 18 on "network design folklore", which is also available online at http://www.informit.com/articles/article.aspx?p=20482
- Gerard J. Holzmann: *Design and Validation of Computer Protocols*. Prentice Hall, 1991, ISBN 0-13-539925-4. Also available online at http://spinroot.com/spin/Doc/Book91.html
- Douglas E. Comer (2000). *Internetworking with TCP/IP - Principles, Protocols and Architecture* (4th ed.). Prentice Hall. ISBN 0-13-018380-6. In particular Ch.11 Protocol layering. Also has a RFC guide and a Glossary of Internetworking Terms and Abbreviations.
- Internet Engineering Task Force abbr. IETF (1989): *RFC1122, Requirements for Internet Hosts -- Communication Layers, R. Braden (ed.)*, Available online at http://tools.ietf.org/html/rfc1122. Describes TCP/IP to the implementors of protocolsoftware. In particular the introduction gives an overview of the design goals of the suite.
- M. Ben-Ari (1982): *Principles of concurrent programming* 10th Print. Prentice Hall International, ISBN 0-13-701078-8.
- C.A.R. Hoare (1985): *Communicating sequential processes* 10th Print. Prentice Hall International, ISBN 0-13-153271-5. Available online via http://www.usingcsp.com
- R.D. Tennent (1981): *Principles of programming languages* 10th Print. Prentice Hall International, ISBN 0-13-709873-1.
- Brian W Marsden (1986): *Communication network protocols* 2nd Edition. Chartwell Bratt, ISBN 0-86238-106-1.
- Andrew S. Tanenbaum (1984): *Structured computer organization* 10th Print. Prentice Hall International, ISBN 0-13-854605-3.

Further reading

- Radia Perlman, *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols (2nd Edition)*. Addison-Wesley 1999. ISBN 0-201-63448-1. In particular Ch. 18 on "network design folklore".

- Gerard J. Holzmann, *Design and Validation of Computer Protocols*. Prentice Hall, 1991. ISBN 0-13-539925-4. Also available online at http://spinroot.com/spin/Doc/Book91.html

External links

- Javvin's Protocol Dictionary (http://www.javvin.com/protocolsuite.html)
- Overview of protocols in telecontrol field with OSI Reference Model (http://www.ipcomm.de/protocols_en.html)
- List of Data Communication Protocols (http://www.zframez.com/protocolsuite.html)
- PDF-Chart showing the Protocols and the OSI reference layer (http://www.draware.dk/fileadmin/WildPackets/Basic/WP_encapsulation_chart.pdf)
- Blog to discuss ideas about modeling and testing of communication protocols (http://blog.protocolbench.org)

Categories: Communications protocols | Data transmission | Network protocols

---