J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering*

HIGH-SPEED PROTOCOLS

Buzzwords such as data highways and information society are becoming ubiquitous and are no longer specific to the research environment. In this context, many emerging applications are pushing the communication world to drastic changes. Currently the most prominent example can be seen in the World Wide Web, *WWW*. Furthermore, the popularity of networked multimedia applications, such as teleconferencing and telecollaboration, is also constantly increasing. As a result, many new opportunities for network users appear in the public sector as well as in the business and commercial sector.

However, in order to serve all these applications, suitable communication systems are required, including the underlying network as well as network and transport layer protocols. All components together need to provide high performance with respect to throughput and latency. Moreover, the integration of multiple services as it is typical for multimedia applications (e.g., audio, video, and data stream) forms a key requirement.

Due to the emergency of fiber-based technology, high-speed networks are being established that enable data rates well over the megabit—even the gigabit—per second threshold. In addition, *ATM* (asynchronous transfer mode) is under development and will be capable of integrating various services within a single network, the *B-ISDN* (broadband integrated services network). Moreover, high-speed protocols and efficient implementation techniques have been developed during the last couple of years. They especially address the specific characteristics of high-speed networks and new application requirements. This article focusses on issues related to high-speed networks.

This article is structured as follows. Important characteristics of high-speed networks are presented first, followed by information on light-weight transport protocols. Protocol mechanisms as well as the most popular light-weight protocols are discussed. A section is devoted to the evolution of the widely used Internet protocol *TCP* (Transmission Control Protocol). Following this, implementation techniques including parallel protocol processing and dedicated hardware support are presented. Finally, some conclusions and perspectives on future trends are given.

Characteristics of High-Speed Networks

High-speed networks are characterized by a high data rate, typically well in the hundreds of Mbps or even in the gigabit per second range and above. However, there is no specific data rate that qualifies a network to be called high-speed network. Compared to traditional low-speed networks (e.g., Ethernet), the data rate is higher by several orders of magnitude. This leads to some very different characteristics of such networks, especially with respect to end-to-end latency. In low-speed networks, end-to-end latency is dominated by the data rate of the link. In contrast to this, the speed of signal propagation clearly dominates the end-to-end latency in high-speed networks. As a result, the so-called *bandwidth-delay-product* is rapidly increasing in high-speed networks. This means that a large amount of data can be buffered within the network. This buffering capability is commonly referred to as *path capacity.*

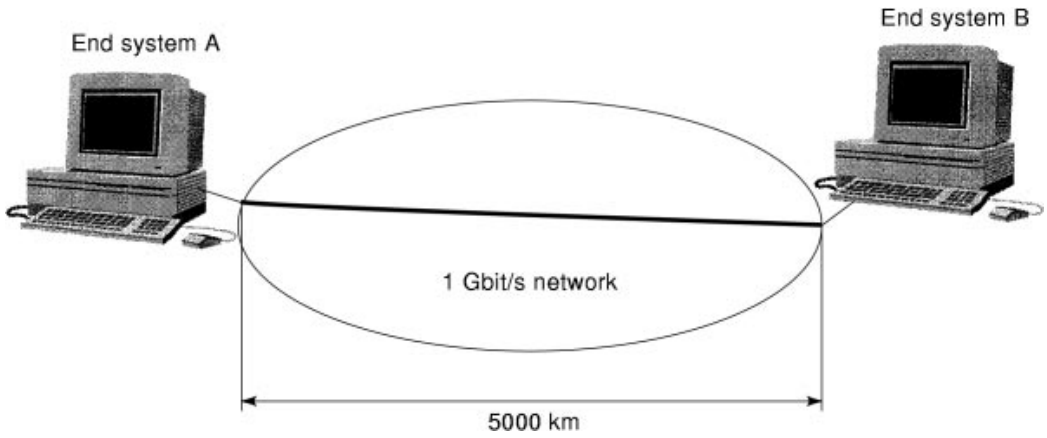

Fig. 1. End systems communicating over a high-speed network.

Table 1. Network Path Capacities (Path = 5000 km)

Network Type	Data Rate r	Path Capacity p
ISDN	64 kbit/s	1.6 kbit
Ethernet	10 Mbit/s	250 kbit
B-ISDN	155 Mbit/s	3.875 Mbit
B-ISDN	1.2 Gbit/s	30 Mbit

The basic scenario depicted in Fig. 1 is used to clarify the importance of the path capacity. Two end systems are interconnected via a communication link of length $l = 5000$ km. The data rate on the link is $r = 1.2$ Gbit/s. Furthermore, we assume the communication link to be a fiber link. The speed of light in a fiber link is approximately $v = 2 * 10^5$ km/s. Thus, the signal propagation speed per kilometer calculates to $\Delta = 1/v = 5$ μs/km. The end-to-end transmission delay d of the communication link is $d = l * \Delta = 25$ ms, that is, the round trip time for end-to-end communication is 50 ms. The path capacity p of a link can be calculated as follows: $p = r * l/v$. For the example, the path capacity is $p = 30$ Mbit, that is, 30 Mbit of data are stored on the transmission link. Table 1 presents path capacities for various networks with different data rates.

The drastic increase in path capacity compared to low-speed networks has a major impact on higher-layer protocols. Several protocol mechanisms that regulate the data flow between end systems are affected, especially at the transport layer. Taking the numbers above, a sending station has to wait 50 ms before it can expect to receive an indication from the receiving station about the transmitted data. However, during that time, the sender can already transmit 60 Mbit (i.e., 7.5 MByte) of data. Thus, it can send a complete file without receiving any indication of proper reception or of any errors. This situation is very different from low-speed networks where usually the first feedback information arrives at the sender after the transmission of a few bits (e.g., 4 bits if ISDN is used in the example). This drastic change of behavior requires enhanced protocol mechanisms.

Moreover, due to the increased speed within the network, the time in which a data unit needs to be handled in the attached systems (end systems, routers, ...) decreases dramatically. Given the previous example, data units of 8 kbytes need to be processed in about 55 μs. However, if the length of the data unit is only 53 bytes (e.g., an ATM cell), it needs to be received and processed in less than 0.4 μs. Comparable numbers in a 10 Mbit/s Ethernet are 6.5 ms and 42 μs, respectively. These numbers underline that requirements on protocol

processing and memory access speed are other significant factors that need to be addressed with the advent of high-speed networks.

Light-Weight Transport Protocols

Protocol mechanisms at the transport layer must address the increasing path capacity in order to provide efficient communication services to the application. In the following, first basic protocol mechanisms are discussed, followed by the presentation of selected light-weight protocols that introduced and applied novel protocol mechanisms.

Basic Protocol Mechanisms. Several protocol mechanisms are typically part of connection-oriented transport protocols that provide a reliable service. Among them are mechanisms for:

- connection establishment and termination
- error control
- flow control

Connection establishment can be seen as a performance critical protocol function, especially with the event of applications that are based on the client/server paradigm. Typically, a handshake-based mechanism is used to establish a connection as depicted in Fig. 2(a). The sender issues a connect request message and needs to wait for a connect indication message before it is allowed to transfer user data to the peer entity. During this handshake procedure some parameters, such as data rate and window size, can be negotiated among the sender, the receiver, and the service provider. This includes *QoS* (quality of service) parameters for multimedia services. However, the handshake procedures lead to a latency of at least one round-trip time before the first byte of user data can be sent, that is, a delay of 50 ms with respect to the example presented previously. In this time 60 Mbit of user data could have been transmitted. Many client/server based applications do not require a large amount of data to be sent and, thus, the time needed for connection establishment, t_{conn}, can easily dominate the time needed for user data transfer, t_{data}, that is, $t_{\text{conn}} \geq t_{\text{data}}$. In order to increase the efficiency of connection establishments, implicit mechanisms have been developed. User data can be transmitted immediately with or after the connection establishment message; see Fig. 2(b). Implicit mechanisms drastically decrease the connection set-up latency. Therefore, transactions can be finalized much faster. For example, the transaction time t_{trans} can be reduced to $t_{\text{trans}} \approx 50$ ms instead of $t_{\text{trans}} > 100$ ms with the handshake-based mechanism. However, guarantees with respect to quality of service can not be given. Therefore, such mechanisms are not targeted for multimedia applications. Protocols serving multimedia applications typically use a separate signaling protocol in order to establish a connection with dedicated QoS requirements.

Connection termination also needs special attention in the environment of client/server applications and high-speed networks. If handshake-based mechanisms are used for establishment and termination of connections, the number of packets and the time consumed for connection management may be higher than the number of packets and the time needed to transmit the user data. Therefore, the number of connection management packets should be minimized. In addition to an implicit connection establishment, so-called timer-based mechanisms have been introduced in order to avoid connection termination messages. Timers are used at both sides to determine the point in time at which the connection is terminated. The timers are updated each time user data are either received or sent. The value of the timer needs to consider the round-trip time of the connection and possible data retransmissions. The mechanism is very sensitive to a proper dimensioning of the timer. A disadvantage of timer-based connection handling can be seen in the fact that connection state must be held longer, since the timer needs to be set to a high enough value. This can be a burden at servers which are frequently requested.

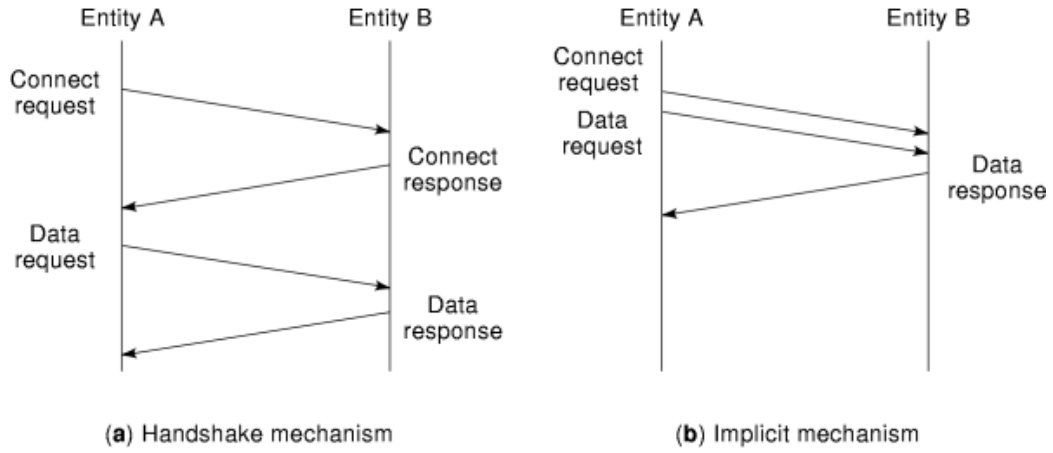

Fig. 2. Connection establishment.

Protocol mechanisms that implement *error control* also need to be adapted for usage in networks with high-path capacity. Typically, very simple error control mechanisms are implemented within transport protocols. Acknowledgments are used by the receiver to indicate the correct reception of data units that have been sent. If a data unit is not acknowledged it is retransmitted by the sender after the timeout of a related timer. The most simple acknowledgment mechanism uses cumulative acknowledgments. All correctly received data units in sequence are acknowledged. All data received subsequently to a corrupted data unit are discarded and not acknowledged. The sender needs to wait for at least a round-trip time in order to know whether the data unit has been received correctly or not.

For retransmission of erroneous data, often the so-called go-back-*N* mechanism is applied, especially in conjunction with cumulative acknowledgments. With this mechanism, all data following the corrupted data unit are retransmitted. Due to the potentially large amount of data in transit, this can dramatically increase the load on the network. In case of selective acknowledgments, more advanced retransmission mechanisms can be implemented that reduce the traffic with respect to the amount of data to be retransmitted. Selective acknowledgments allow—in contrast to cumulative mechanisms—the acknowledgment of data that has been correctly received subsequent to a corrupted data unit. The disadvantage can be seen in increased memory requirements at the receiving system. However, latency can be significantly reduced which is highly desirable for many applications, especially for interactive applications (e.g., teleconferencing). Selective acknowledgments are very attractive in high-speed networks since they do not require the retransmission of all data in transit. Furthermore, forward error correction (*FEC*) appears as a useful alternative in high speed networks since it completely avoids the round trip time to correct—at least part of—the errors. The basic principle is that redundant data are transmitted. Thus, the receiver may be capable to reconstruct the original data even in case of lost or corrupted data. However, this does not guarantee complete reliability. Therefore an additional option to use retransmissions is required. This is needed if too many data are corrupted or lost and, thus, the original data cannot be reconstructed. The potential disadvantages of FEC can be seen in the increased load on the network as well as in the considerable high processing power needed for good FEC mechanisms.

The task of *flow control* is to regulate the flow of data between two communicating protocol entities. The resources of the receiving system (buffer, processing power) are protected against overload conditions. The basic mechanism used in various transport protocols is a credit-based sliding window mechanism. The sender has a certain credit—the window—that is, an amount of data (measured in bytes or data units) that it is allowed to send before receiving an acknowledgment. This credit basically reflects the buffer capability of

the receiver. With increasing path capacity the utilization of the communication link by two communicating stations decreases with such an approach. Typical window sizes are in the range of kbytes and, thus, smaller than the path capacity of high-speed networks. Therefore, a new mechanism called *rate control* has been developed during the last couple of years. It is applied in various light-weight protocols. With rate control, traffic is controlled via the rate of the sending station in contrast to the credit used by sliding window. Sending and receiving system as well as the network agree on this rate, that is, the receiving system and the network claim that they can process data received with the rate agreed upon.

Light-Weight Transport Protocols. During the past twenty years various transport protocols have been developed. The Internet protocol TCP can be stated as the most prominent example. It was one of the earliest transport protocols and currently is implemented on almost all computers. Since many changes in the networks can be observed, especially with the event of high-speed networks, the requirements on transport protocols have changed. Furthermore, application requirements did dramatically change. The client/server-paradigm is penetrating the communication area today and service integration is getting increasingly important with respect to multimedia applications.

In order to address the changes related to high speed networks, various so-called light-weight transport protocols have been developed. Light-weight protocols shorten the regular data path and, generally, minimize protocol overhead. The most prominent ones are briefly described in the following with respect to their individual contributions. The protocols are:

- Delta-t (1), (2)
- NETBLT (3), (4)
- VMTP (5), (6), (7)
- URP (8)
- XTP (9), (10)

The "father" of transport protocols can be seen in TCP. Each of the protocols listed above inherits some mechanisms from TCP and replaces or improves other mechanisms. An excellent overview of the correlation among these protocols can be found in (11). The transport protocol of the OSI protocol suite, transport protocol class 4 (*TP4*), is not discussed further since it does not introduce any new protocol mechanisms and it is not of practical relevance today. TCP itself is constantly undergoing changes. Those relevant to high-speed networks are summarized in the section on TCP.

The presented light-weight transport protocols have been designed with a connectionless network service in mind, such as *IP* (Internet Protocol). The only exception is URP which is based on the virtual-circuit oriented network Datakit.

Delta-t. In the late 1970s, the Lawrence Livermore Laboratory designed a new transport protocol called Delta-t. The development was driven by the idea to provide high performance communication support for client/server applications. Delta-t presents one of the earliest protocols that addresses the specific requirements for client/server applications. It provides some trail-blazing approaches that are more or less standard today. One of its main characteristics is that it minimized the latency between initiating the connection establishment and the actual start of the data-transfer phase as well as the overall number of data units exchanged. The novelty that is introduced by Delta-t is an *implicit connection establishment mechanism*. It minimizes the number of data units exchanged and, thus, the delay until user-data transfer can start. Handshake procedures are completely avoided. Data can be transferred in Delta-t immediately after issuing the connection establishment. Another novelty of Delta-t is the definition of a *timer-based protocol mechanism* for connection termination. It avoids a handshake procedure for connection termination and, thus, further reduces the overhead with respect to the number of data units exchanged. For such a mechanism, unambiguous connection identifiers are needed. Furthermore, connection identifiers need to be frozen after connection termination, that

is, they cannot be immediately used by a newly established connection, in order to avoid problems with delayed duplicates of an already closed connection.

Although Delta-t is not used widely today, it provided important insights in efficient support of client/server applications. The connection handling mechanisms have been adopted by subsequent protocols, even by TCP (see section on TCP).

Network Block Transfer Protocol. The protocol *NETBLT* (network block transfer protocol) was developed at MIT. It directly targets transmission links with high-path capacity. However, it addresses links with a low-speed data rate: satellite links. The extreme length of satellite links (72000 km) leads to a high-path capacity without the necessity of a very high data rate on that link. In such environments, window based flow control can not be applied efficiently. It either leads to large windows or underutilized communication links. NETBLT specifically addresses this problem and a novel mechanism that combines window based flow control with a newly introduced *rate based flow control* is presented. Furthermore, in NETBLT error control and flow control are decoupled and, thus, not overloaded.

NETBLT introduces so-called *bursts* for the implementation of the rate based flow control. Two burst parameters are defined to regulate data transmission: burst size and burst rate. The burst size limits the length of a burst, that is, the amount of data that can be sent continuously. The burst rate defines the minimum time interval between two subsequent bursts.

Consequently, the data rate on the link is controlled by these two parameters. Therefore, larger windows can be used without causing an overflow at the receiver. Burst size and burst rate are negotiated among the involved parties.

With respect to error control, NETBLT also introduces novel mechanisms. It uses a combination of cumulative and selective acknowledgments. *Selective acknowledgments* signal that certain data units have been lost or corrupted. Cumulative acknowledgments signal the correct reception of a sequence of data units. Since selective acknowledgments are used, NETBLT can apply selective retransmissions as well. In order to reduce the overhead introduced by acknowledgments, so-called buffers are introduced as basic units for the handling of acknowledgments. Buffers contain a number of data units that are logically treated as single unit. Thus, an acknowledgement is needed with respect to buffers and not according to the number of data units exchanged. The size of the buffer is negotiated between sender and receiver.

Versatile Message Transaction Protocol. The protocol *VMTP* (versatile message transaction protocol) was developed at Stanford University for usage with the distributed operating system V. The main purpose was in an efficient support of remote procedure calls, that is, transaction-based applications. The goal is somewhat comparable to that of Delta-t.

VMTP introduces so-called *message transactions* that model remote procedure calls. A message transaction comprises the request and one or multiple optional response messages. Message transactions can be directly forwarded from one server to another without involvement of the client. The design of VMTP incorporates an implicit and timer-based connection handling derived from Delta-t.

Furthermore, VMTP supports *multicasting,* that is, a transaction request can be sent to multiple servers. It is among the first protocols that address the need for specific multicasting support. A novelty of VMTP can be seen in the 64-bit long *unambiguous connection identifiers* which are independent from the network layer address. This enables the migration of VMTP entities in the network, for example in order to implement load balancing in distributed systems. VMTP further applies a rate control mechanism in order to control the data flow between communicating entities. The mechanism is comparable to NETBLT. However, the *rate control* is also slightly different from that applied in NETBLT. VMTP uses so-called interpacket gaps, that is, it controls the gaps between two consecutive data units. Therefore, a finer timer granularity is required compared to the interburst gaps controlled in NETBLT. However, VMTP was from the beginning developed with a hardware supported implementation in mind. Thus, a higher timer resolution could be integrated more easily than within a software implementation.

The idea of a hardware implementation also influenced the packet format defined in VMTP. A pipelined processing of different header fields (e.g., encryption, checksum) is directly supported.

Universal Receiver Protocol. The protocol *URP* (universal receiver protocol) was developed in the mid-1980s at AT&T in order to provide universal data transport across their Datakit network.

The communication structure of URP is based on *simplex transmissions,* thus, a full duplex connection is composed of two simplex connections, one per direction. Data are handled on the bases of 8 bits. Two operating modes are distinguished for the receiver: block mode and character mode. In *block mode,* sequence numbered blocks are used for data transmission. Block retransmission is available as an option in case of error. In *character mode,* data are transmitted as character stream without the option of retransmission. The service provided by character mode and by block mode without retransmission is very different from the services discussed previously. Data are delivered error-free and in sequence, but some data in between may be lost. Moreover, in character mode, data losses can not be detected. Thus, character mode reflects a very simple service that can be implemented with low overhead. Generally, URP is one of the first protocols that provides some flexibility with respect to the provided service. This is very important, especially with the advent of multimedia applications. The grade of URP service can be selected by the transmitter.

Since URP was designed for high-speed networks, optimizations were integrated in the protocol. One of the most interesting design issues is the relocation of some processing intensive tasks from the receiver to the sender. This is motivated by the fact that usually the receiver represents the performance bottleneck. Protocol mechanisms that follow such an approach are called *sender-driven mechanisms.* This is reflected mainly in the acknowledgment procedure of URP. The sender explicitly asks for an acknowledgment from the receiver, for example, after having sent a full window of data. Thus, the decision and the processing related to it are located on the sender side and not at the receiver side. Furthermore, in block mode a so-called reject mechanism is implemented that basically reflects a selective acknowledgment. This allows the speed up of retransmissions. The data format used in URPs block mode is very different from those of other protocols. It uses a trailer of 4 bytes that follows a block of data; no header is involved. This simplifies processing in hardware. A hardware based implementation of URP was a vision of the protocol designers.

URP is not widely used today. However, many concepts are inherited and enhanced in XTP which is among the most prominent protocols during the last couple of years.

EXpress Transport Protocol. The protocol *XTP* (express transport protocol) was developed with the goal of a VLSI implementation. Moreover, it was targeted toward an efficient support for real time applications.

The idea of a hardware implementation influenced the data format of XTP similarly to VMTP and URP. However, it needs to be stated that the current data format of XTP 4.0 is very different from that described in the first specifications of XTP. For example, the control fields moved entirely from the packet trailer to the header.

XTP introduces an *out-of-band signaling* at the transport layer. It is the first protocol that uses out-of-band signaling over connectionless network services. The exchange of control information between XTP entities and the transfer of user data are clearly separated. Moreover, the protocol was designed as a set of orthogonal protocol functions that can be mapped onto a multiprocessor platform. Such a design also forms a sound basis for a protocol that provodes multiple services. This idea was inherited from URP. XTP provides many different mechanisms and, thus, is capable of supporting different services for a large variety of applications. Different strategies and options of protocol mechanisms can be selected by individual applications. Generally, XTP uses many of the protocol mechanisms introduced by earlier protocols, for example, selective acknowledgments and rate-based flow control. Moreover, the use of sender-driven mechanisms of URP which relaxes processing requirements at the receiver is applied. In some sense, XTP collected mechanisms that are suited for high-speed networks from earlier protocols.

XTP introduces a flexible addressing mechanism, that is, different addressing schemes can be applied (e.g., Internet addressing or OSI addressing). With this flexibility, XTP can be used in different networking environments, such as Internet or OSI. However, today it is mainly used over the Internet, that is, over IP.

Furthermore, XTP provides multicasting. Different mechanisms have been supported over time. Multicast support forms an important issue for many emerging applications. Proper support is still under discussion today. Version 4.0 provides a reliable multicast service.

Evolution of TCP

As discussed in the previous section, many new transport protocols have been designed with respect to the characteristics of high-speed networks and with respect to application requirements. However, none of these protocols is widely used today. TCP, which was developed over twenty years ago, clearly dominates the market. However, TCP has also changed over time; mechanisms have been integrated to solve some problems of TCP and that support TCP over high-speed networks. Some prominent examples are:

- avoidance of the Silly Window Syndrome
- window scaling
- connection count

TCP uses a sliding window mechanism for flow control and cumulative acknowledgements with go-back-N retransmission. In this context, one of the first performance problems that was observed was the so-called *Silly Window Syndrome.* It is characterized by a situation in which the receiver advertises only small windows and, thus, the sender transmits data in small segments (12). Such a situation is initially caused by a sender, who—in the case of a push flag—has only few data to send which are acknowledged immediately by the receiver. During long data transmissions (e.g., large file transfers) the sender cannot recover from such a situation and the performance of the data transfer can decrease drastically. However, the Silly Window Syndrome can be avoided by using the following simple algorithms at the receiver and sender, respectively. The sender should refrain from sending small segments. As a general rule the sender should only send if the window is filled by more than 25% of the window size. At the receiver side, two complementary algorithms should be implemented. First, the receiver should avoid advertising small windows. The receiver should only offer a new window if at least some fraction of the overall window size (e.g., 50%) can be offered. Furthermore, the receiver should refrain from sending an acknowledgment at all if the push flag was not set in the received segment and no data are flowing in the reverse direction. This decreases the processing requirements of the sender since the number of acknowledgments that it needs to process is reduced. These algorithms are suggested for use with TCP in the host requirements document (13).

TCP uses a 16-bit field for window advertisements and the basic unit for advertisements is a byte. Thus, the *maximum window size* $w_{\max}$ can be calculated as follows: $w_{\max} = 2^{16}$ byte $= 65$ kbyte. With this limited window size, high-speed networks cannot be highly utilized. In order to overcome this problem, a new option has been introduced in TCP that allows the use of other basic units for window scaling. The basic unit is advertised within a SYN segment during connection establishment (see Fig. 3). The basic units are multiples of a byte and they are coded logarithmically, for example, the factor 3 leads to a scaling factor $S = 2^3 = 8$. There exists an upper limit of $S = 2^{14}$ for the scaling factor in order to bound the highest usable sequence number.

Proper support of client/server applications leads to another new TCP option, the so-called *connection count.* Connection counters are used to avoid the handshake procedure for connection establishment. This version of TCP is called *T/TCP* (transaction TCP) or *TAO* (TCP accelerated open). The connection count is monotonically increased every time a new connection is established. The server stores the last connection count received from a client. If a SYN segment is received by the server, it compares the connection count with the value stored at the server. If the received connection count is larger, the connection can immediately be established. If it is smaller, the normal handshake procedure is used.

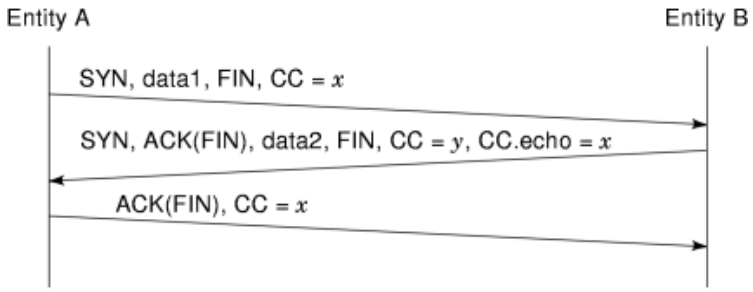

Fig. 3. Transaction support of T/TCP.

Implementation Techniques

During the past few years, implementation techniques for communication systems have been discussed for various reasons. The two main reasons are efficiency and flexibility. The latter occurs especially in environments with networked multimedia applications and is not discussed further here.

Given that currently available transmission technology can deliver data across networks at rates exceeding the gigabit per second range, one challenge in building high performance communication systems is the design of efficient protocol implementations. Considering the increasing gap between the growth of physical network bandwidth and the growth in processing power available for protocol processing, multiprocessor platforms were considered to overcome performance bottlenecks. Therefore, the communications system has to be adequately partitioned in order to make use of such architectures.

Besides processing requirements, data manipulation functions (i.e., memory access and data copies) play a crucial role for high performance communication systems. The technique of integrated layer processing has been introduced to overcome this bottleneck. Due to the many difficulties with this technique, no major breakthrough could be achieved up to now.

Parallel Protocol Processing. In this section, different types and levels of parallelism that can be applied in communication systems are introduced first. Then, some of the most prominent projects in parallel protocol processing are briefly summarized and their results are evaluated.

Types of Parallelism. Two types of parallelism can be distinguished:

- spatial parallelism
- temporal parallelism

Spatial parallelism is based on the mapping of the processing task on multiple concurrently operating processing units. Two types of spatial parallelism can be distinguished:

- SIMD-like organization
- MISD-like organization

Spatial parallelism based on an *SIMD* (single instruction multiple data) organization reflects parallelism among identical tasks independently and concurrently operating on different data units. A scheduling discipline, for example, round robin, may be used to allocate data to processing units. An SIMD organization requires only a minimum of synchronization among the processing units. However, it does not decrease the processing time for a single data unit. However, it increases the number of data units processed during a certain time interval. The performance benefits are limited by the maximum number of concurrently utilizable

processing units which can be approximately calculated by multiplying the mean packet processing time with the target throughput.

Spatial parallelism based on an *MISD* (multiple instructions single data) organization is associated with concurrent processing of different tasks on the same data. Applying an MISD organization reduces the processing time required for a single data unit since multiple tasks are processed concurrently. However, a high degree on synchronization among the processing units may be required. Careful balancing of the system is needed to ensure that synchronization overhead does not paralyze system performance.

Temporal parallelism is provided by the concept of pipelining which is similar to assembly lines in industrial plants. To achieve pipelining, the processing task has to be subdivided into a sequence of subtasks each mapped on a different pipeline stage. Thereby, each pipeline stage processes different data at the same point in time. The protocol VMTP has especially addressed this issue in the protocol design. Pipelining does not decrease the processing time needed for a single data unit. It increases the completion rate of packets and, consequently, the system throughput. Since the performance of a pipeline is limited by its slowest stage, stage balancing is an important issue.

Levels of Parallelism. In communication subsystems, mainly four different levels of parallelism can be distinguished based on the granularity of an atomic unit:

- stack level
- entity level
- function level
- intra-function level

The *stack level* forms the most coarse-grained level of parallelism applied to communication subsystems. An atomic unit comprises a complete protocol stack consisting of different protocol layers. Mainly spatial parallelism providing SIMD organization can be implemented at the stack level. Each stack can be associated with a separate connection or a dedicated application data stream (e.g., in case of connectionless services). In this case, packets are scheduled on a *per connection* basis and parallelism takes place among concurrent connections. However, this approach does not improve the performance of a single stack and, moreover, load balancing between the stacks cannot be influenced since the relation of packets to corresponding connections dictates the data distribution over the processing units. As a consequence, some processing units may be heavily loaded requiring additional processing power, whereas others are lightly loaded or even idle. An alternative approach using *per packet scheduling* may lead to a more beneficial load distribution. Packets are scheduled based on the availability of processing units, independent of their relation to a connection, or an application data stream. The performance of a single connection can be increased by concurrently processing multiple data units associated with a single connection. However, this solution requires synchronization among processing units concurrently operating on different packets belonging to the same connection.

The *entity level* provides a finer granularity. An atomic unit is either associated with a complete protocol, or, more modular, with the receive entity formed by the receive part of the protocol, and the send entity formed by the send part. Spatial parallelism can be applied resulting in the same trade-offs as at the stack level. Entities belonging to different protocol layers can be implemented in a layer pipeline using temporal parallelism to form a complete protocol stack. The layer pipeline can further be subdivided into a receive pipeline and a send pipeline which are mutually independent to a large extent. They may coordinate their operation on data units belonging to the same connection by using common connection control information. Furthermore, synchronization among send and receive pipeline may be required.

At the *function level,* protocol functions are used as atomic units (e.g., connection establishment, flow control). A first and necessary step in that direction is the analysis of protocols in view of extracting the intrinsic parallelism among protocol functions. The result of this analysis can be given in the form of a dependency graph representing the relationship among the protocol functions, mainly in terms of data dependency.

At the *intra-function level,* concurrent processing is applied in order to increase the performance of a single protocol function. Some examples include computing intensive functions, such as checksum processing, encoding or encryption, or functions operating on large data bases, for example, the routing function.

Experiences in Parallel Protocol Processing. Many projects on parallel protocol processing had been started by the end of the 1980s. In this article, only a few are selected that provide dedicated experience and cover the main areas just presented. Most projects experiment with the OSI protocol stack. Few projects address new protocols, such as XTP.

One project intensively investigated parallelism at the stack level (18). Important issues that need to be considered at that level are packet ordering, handling of segmented data, access to connection control information, and shared protocol data (e.g., IP routing table). Packets are scheduled on a packet-by-packet bases among the involved stacks. In (19) a centralized internal sequence number scheme is applied to packets as well as to control information. Based on this, processing of packets or control information arriving out-of-order is postponed until a processor identifies them as the next to be processed. To avoid any synchronization problems associated with concurrent processing of connection control information, a separate disjoint processing stage is implemented. This is a proper design in cases where packet processing dominates control processing, for example in the presentation layer. Since scheduling is performed on a per packet bases, segments of the same data unit may be processed by different processing units. The reason is that during reception from the network it cannot be determined whether a segment of some higher layer packet is received. This problem of segmented data units can be avoided by using per connection scheduling at the stack level. However, this requires an unambiguous identification of higher layer connections during packet reception from the network, that is, *intelligent packet filters* are needed. The results reported in (18) were achieved with up to four processors. A performance improvement of up to 3.5 was achieved with four processors. The results were highly dependent on the size of a data unit. Large data units allow for a higher degree of parallelism and, thus, lead to better performance.

Few projects addressed *function level parallelism.* Some of them with dedicated HW support (20,21), some of them using transputer networks (22,23). The XTP project even tried to directly couple protocol design and VLSI (very large scale integration) implementation (24). They failed with respect to VLSI implementation. However, XTP became a very prominent protocol that highly influenced the development of communication systems over the last years, especially with respect to integrated services and multicasting.

The projects that were using *transputer networks* for parallel protocol processing either applied function level parallelism (15,22) or entity level parallelism (25). All of them needed to overcome the distributed memory architecture associated with transputer networks. In (25) a global memory for several transputers was implemented. Generally, these projects achieved some performance advantage compared to traditional implementations. However, with respect to the increased cost due to multiprocessing, these numbers are not very promising. This mismatch between cost increase and performance speed-up can be seen as one of the main reasons why parallel protocol processing did not succeed up to now. In (25) a speed-up of 1.55 is reported with 2 transputers and a speed-up of 2.17 with the use of four transputers. The reason for the low speed-up numbers can be found in the uneven loadbalancing that occurs because the protocols were not designed for parallel processing. In (22) a speed-up of 3.73 was reported with the use of eight transputers. Again, uneven load balancing is the reason for this comparable low speed-up.

In Ref. 26 a protocol processor consisting of multiple processors and memory modules was applied for the implementation of a light-weight transport protocol. The main characteristic of the protocol can be seen in the exchange of complete state information instead of state updates. Processing of 20,000 headers/s is expected to be feasible. The operating system was identified as performance bottleneck.

Located between entity level parallelism and function level parallelism are approaches that accelerate processing of an entity by the use of dedicated hardware for some performance critical functions, such as buffer management, timer management and the like. In (27) a dedicated software process was in charge of buffer management. The XTP project, for example, targeted a dedicated VLSI support of buffer management.

Furthermore, the usage of specialized memory chips, such as CAMs, was addressed in (20) in order to speed-up timer management. For the implementation of the retransmission timer, CAMs were used. However, the usage of special memory did not succeed mainly because of cost reasons. Checksumming is a typical protocol function that can be implemented in hardware. In transport protocols it usually requires read access to the user data along with some arithmetical calculations. Dependent on the location of the checksum in the data unit, this function can be processed on-the-fly during data movements applying pipelined parallelism and reducing the number of memory accesses needed. Advanced network interfaces provide support for on-the-fly processing of transport layer checksums on the network adapter. Pipelining is usually applied at the interface between network adapter and host processing. Efficient implementation techniques are required to avoid this interface becoming the system bottleneck. Mapping of the adapter memory into the hosts virtual memory is a possible way of avoiding data movements between host and adapter. Examples for memory-mapped network interfaces are (28) and (29).

Summary

The event of high-speed networks with a high path capacity stimulated the development of so-called light-weight transport protocols. This article provided an overview of the most important light-weight protocols and the novelties that were introduced by these protocols. Moreover, the evolution of TCP is briefly summarized with respect to high-speed networks. It can be observed that some of the mechanisms introduced within the context of light-weight protocols are now being used in TCP. Examples are implicit connection establishment and timer-based connection termination. Thus, it can be expected that TCP will further adapt promising mechanisms if they are required by dedicated applications. An example that is under discussion is the support of selective acknowledgments. Furthermore, it can be expected that TCP will stay among the most popular protocols—it will not be replaced by a light-weight transport protocol. However, with respect to multicasting, it might be replaced by transport protocols that have dedicated multicasting support. An excellent investigation on protocol mechanisms for reliable multicast protocols can be found in Ref. 30.

The increased network speed also imposes higher processing requirements on communication systems. Parallel processing can be seen as one approach to increase processing power. The possibilities of applying parallelism to protocol processing have been briefly summarized. Some of the most interesting projects have been selected and presented with their main achievements. However, it must be stated that, generally, parallel protocol processing did not succeed for several reasons. Protocols are not designed for parallel processing which leads to problems in load balancing among the processors involved. Thus, no linear speed-up can be achieved. This leads to the problem that the increased cost due to parallel processing cannot be justified. The most promising approach can be seen in entity acceleration by implementing dedicated functions in hardware. Implementations complient with this approach can be found quite often (e.g., hardware-based on-the-fly checksumming). Finally, it needs to be stated that even entity acceleration may be less important for certain protocol functions in the future. This is because processors do provide more and more dedicated instructions for performance intensive functions related to multimedia (e.g., video coding (28)).

BIBLIOGRAPHY

1. R. W. Watson *Timer-Based Mechanisms in Reliable Transport Protocol Connection Management*, Computer Networks, North-Holland, Vol. 5, 1981, 47–56.
2. R. W. Watson The Delta-t Transport Protocol: Features and Experience, in H. Rudin, R. Williamson (eds.), *Protocols for High-Speed Networks*, Elsevier (North-Holland), 1989, pp. 3–18.
3. D. Clark M. Lambert L. Zhang NETBLT: A bulk data transfer protocol, Request for Comments RFC 998, March 1987.

4. D. Clark M. Lambert L. Zhang NETBLT: a high throughput transport protocol, Proceedings of the *ACM SIGCOMM '86*, Stowe, VT, 1986, 353–359.
5. D. Cheriton VMTP: A protocol for the next generation of communication systems, *ACM SIGCOMM '86*, Stowe, VT, pp. 406–415.
6. D. Cheriton C. Williamson VMTP as the transport layer for high-performance distributed systems, *IEEE Commun. Mag.*, **27**: June 1989, 37–44.
7. D. Cheriton C. Williamson VMTP: versatile message transaction protocol–protocol specification, Request for Comments RFC 1045, February 1988.
8. A. G. Fraser W. T. Marshall Data transport in a byte-stream network, *IEEE J. Selected Areas in Communications*, 7: September 1989, 1020–1033.
9. W. Strayer B. Dempsey A. Weaver *XTP. The Xpress Transfer Protocol*, Reading, MA: Addison-Wesley, 1992.
10. XTP Forum, Xpress transport protocol specification revision 4.0, Technical Report, March 1995.
11. W. Doeringer *et al.* A survey of light-weight transport protocols for high-speed networks, *IEEE Trans. Commun.*, **38**: 1990, 2025–2039.
12. D. Clark Window and acknowledgement strategy in TCP, Request for Comments RFC 813, July 1982.
13. R. Braden (Ed.) Requirements for internet hosts—communication layers, Request for Comments RFC 1122, October 1989.
14. W. Stevens *TCP/IP Illustrated*, Volume 3, Reading, MA: Addison-Wesley, 1996.
15. M. Zitterbart High speed transport components, *IEEE Network Magazine*, January 1991, 54–61.
16. D. Clark D. Tennenhouse Architectural considerations for a new generation of protocols, *ACM SIGCOMM '90*, September 1990, 200–208.
17. B. Ahlgren M. Bjorkman P. Gunningberg Integrated Layer Processing Can Be Hazardous to Your Performance, in W. Dabbous, C. Diot (eds.), *Protocols for High-Speed Networks*, V, London: Chapman & Hall, 1996.
18. R. Lam M. R. Ito On the parallel implementation of OSI protocols processing systems, Proceedings IPPS94, Cancun, Mexico, April 1994.
19. M. Goldberg G. Neufeld M. Ito A Parallel Approach to OSI Connection-Oriented Protocols, in B. Pehrson, P. Gunningberg, S. Pink (eds.), *Protocols for High-Speed Networks*, III, Elsevier (North-Holland), 1992, pp. 219–232.
20. D. Feldmeier An Overview of the TP++ Transport Protocol Project; in A. N. Tantawy (ed.), *High Performance Networks: Frontiers and Experience*, Norwell, MA: Kluwer-Academic Publisher, 1993.
21. T. F. La Porta M. Schwartz Performance analysis of MSP: A feature-rich high-speed transport protocol, *IEEE/ACM Trans. Networking*, **1** (6): 740–753, 1993.
22. M. Zitterbart Parallelism in Communication Subsystems, in A. N. Tantawy (ed.), *High Performance Communications*, Norwell, MA: Kluwer-Academic Publisher, 1994, 177–194.
23. T. Braun M. Zitterbart A parallel implementation of XTP on transputers, *16th IEEE Conference on Local Computer Networks*, Minneapolis, MN, October 1991, 91–96.
24. Protocol Engines, *PE-1000 Series*, Protocol Engine Chipset, 1992.
25. M. Kaiserswerth The parallel protocol engine, *IEEE/ACM Trans. Networking*, **1**: 1993, 650–663.
26. A. N. Netravali W. D. Roome K. Sabnani Design and implementation of a high-speed transport protocol, *IEEE Trans. Commun.*, **35** (11): 2010–2024, 1990.
27. M. Kaiserswerth A parallel implementation of the ISO 8802-2.2 protocol, *IEEE TRICOMM '91*, Chapel Hill, NC, April 1991.
28. M. Blumrich *et al.* Virtual-memory-mapped network interface; *IEEE Micro*, February 1995.
29. R. Minnich D. Burns F. Hady The Memory-integrated network interface; *IEEE Micro*, February 1995.
30. D. Towsley J. Kurose S. Pingali A comparison of sender-initiated and receiver-initiated reliable multicast protocols, *IEEE J. Selected Areas Commun.*, **15** (3): 398–406, 1997.
31. A. Peleg S. Wilkie U. Weiser Intel MMX for multimedia PCs, *Communication of the ACM*, **40** (1): January 1997.

MARTINA ZITTERBART
TU Braunschweig