Making XTP Responsive to Real-Time Needs

W. Timothy Strayer, Bert J. Dempsey, Alfred C. Weaver

Computer Science Report No. TR-89-18
December 1, 1989

Making XTP Responsive to Real-Time Needs

W. Timothy Strayer, Bert J. Dempsey, Alfred C. Weaver

Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903
(804) 924-7605
wts4x@virginia.edu, bjd7p@virginia.edu, acw@virginia.edu

ABSTRACT

Conventional transport protocol standards do not adequately support distributed real-time systems. An ideal protocol would ensure that at all times the communications subsystem is working on the task (message) of the most value to the overall system. In this report we investigate how to develop a flexible and potent discrimination scheme based on *importance*, an abstract concept that measures the degree to which a task contributes to the overall system goal. We examine typical communication subsystems for what can and can not be expected from such a policy. We review our chain of reasoning for producing several schemes with varying degrees of complexity and usefulness. Finally, we look at the Xpress Transfer Protocol as a case study of a transport layer protocol in development, and describe and critique the methods proposed for making it responsive to special needs through a discrimination policy based on the importance of the messages it must process.

1. Introduction

The subject of real-time, both in operating systems and communications, has generated much literature and as many definitions as there are systems which claim to be real-time. As an attribute, real-time commonly refers to a system's ability to accomplish its goals in the presence of time constraints. These time constraints usually manifest themselves as deadlines, and separate real-time systems into hard and soft deadlines. A hard real-time system is one for which missing a single deadline is disastrous; a soft real-time system relaxes the constraint to say that missing a deadline degrades the system. As systems become more distributed, the ability for a communications subsystem to provide services which are responsive to real-time needs is more widely demanded.

There are two approaches to endowing a system with real-time capabilities: provide services to system resources which are completed so quickly that the resources never represent a point of contention; or provide the system with a manner of arbitrating between competing tasks for the system resources. The former method is by far the more popular — increasing clock speed has an immediate and tangible effect on a system. However, advances in sheer speed cannot always outpace demands on a system. Furthermore, as systems become more highly loaded, differentiating among tasks to promote the most important activities at the expense of the less important ones becomes essential. Therefore, tasks within the system are usually *prioritized* by some scheme so that these tasks may be ordered while competing for shared resources. A priority is a mechanism by which a task has a relative importance assigned to it for use during competition for resources. This importance, therefore, imposes a ranking among all tasks. The server then chooses the highest priority task each time it can make such a choice. Within a computing environment, this satisfies, at least nominally, the concerns of users that all tasks are not equal.

One of the most valuable resources within a distributed real-time system is the communications subsystem. Information transfer and process synchronization depends on the timely and reliable performance of this subsystem. The communications subsystem must therefore be responsive to the needs of the applications within the system without hindering these applications unnecessarily. By making the underlying network as fast as possible, hopefully much of the contention can be eliminated. Unfortunately, performance measurements of several implementations of ISO standard communications protocols indicate that communications subsystem and the interface to the subsystem are inefficient and use a fraction of the available bandwidth [STRA88a,b,c]. When contention exists, there must be some way to arbitrate between competing tasks in a method that serves the overall purpose of the system. Ranking the tasks via some priority scheme is a reasonable means to provide such arbitration, although a useful number of priorities is still debatable [PEDE88].

Three observations, however, pervade the use of priorities. First is the issue of global knowledge. Unless each transmitting entity is both aware of and adheres to a globally administered priority scheme, then locally determined priorities are meaningless since they cannot fit into the global priority scheme. Second, no communications protocol in use or in development avoids the problem of priority inversion, where a task of lower priority may prevent a task of higher priority from being served. Finally, even if the first two concerns were satisfied, most priority schemes are static, whereas it is more general to consider tasks whose relative importance changes over time according to the changes in the environment. Four observations, however, pervade the use of priorities. First is the recognition that there are possibly as many priority domains as there are layers in the communications subsystem's architecture. A message can be assigned a priority at its entry into the server, which may different from and possibly unrelated to the priority assigned at the Transport layer, which may be different from the priority of the Network layer used in routers, which may also be different from the priority used to access the media. Either there must be a unified priority domain that transcends all layers, or there must be consistent treatment of the priorities at each layer; i.e., all points of contention must discriminate with the same discrimination policy. Second is the issue of global knowledge. Unless each transmitting entity is both aware of and adheres to a globally administered priority scheme, then locally determined priorities are meaningless since they cannot fit into the global priority scheme. Third, no communications protocol in use or in development avoids the problem of priority inversion, where a task of lower priority may prevent a task of higher priority from being served. Finally, even if the other concerns were satisfied, most priority schemes are static, whereas it is more general to consider tasks whose relative importance changes over time according to the changes in the environment.

In this paper we briefly survey the literature on requirements for real-time distributed systems. The literature does not agree, of course, on a canonical set of requirements; we provide this survey to show that there is little agreement in the field about such requirements, and yet some underlying tenets exist, such as the need to be flexible and robust.

We continue by building our own abstraction, the importance abstraction, and relate requirements of a communications subsystem (including real-time requirements) to this abstraction. The importance abstraction is based on the principle that all tasks in a system contribute to the overall system goal, and importance is the measure of their contribution. This is a starting place, where we can make absurd assumptions. It does, however, help us to understand the system concepts and define the roles and responsibilities of the various components of the system and subsystems, especially the communications subsystem.

We then state the goal of this research, which is to find a scheme that can provide a flexible and potent discrimination policy, yet is possible and appropriate for a communications subsystem. We examine typical communications subsystems for what can and cannot be expected from such a policy. This investigation yields an interesting list of open questions and research directions.

We examine some schemes and ideas as possible solutions to the problem of making a communications subsystem responsive to real-time and other special needs. We find we are stymied by the need for such a scheme to be powerful and yet implementable, so it is instructive to include these "first tries" at offering a general scheme, since they have helped us to realize that this problem is genuinely hard.

Finally, we present the scheme adopted by XTP Revision 3.4 for the *sort* field, which addresses the problem of how XTP can avoid the shortcomings of its predecessors inasmuch as real-time communications' needs are concerned. A survey of the attempts in earlier Revisions of the XTP Definition to deal with this issue is followed by an explanation and evaluation of the *sort* field mechanism for Revision 3.4.

2. Requirements for a Real-Time Communications Subsystem

It is important to remember that the communications subsystem is not responsible for the attribute *real-time*; rather it is responsive to the needs of the system, and that includes the special needs of a real-time system. What are the special needs of a real-time system? This question has no universal answer; the term *real-time* is vague and ambiguous. Given the emphasis on designing a communications subsystem to support real-time systems, however, the question deserves discussion.

A real-time distributed system consists of several applications that are distributed among different processing nodes connected via a common network. The applications may be executing in parallel subject to both precedence and timing constraints. The communications subsystem has responsibility to provide the services necessary for a timely response to changes in the application's environment. It must provide sufficient functionality and performance to be useful to applications which require time-constrained communication, allowing the application to effectively handle error conditions without the communication system inducing errors of its own ([STRA88d]).

There are three major requirements of distributed real-time processing on a local area network environment [ZNAT87]. *Robustness* is the combination of reliability, availability, and dependability requirements, and reflects the degree of system insensitivity to errors and misinformation. *Flexibility* relates to the ease of designing and structuring a network that can support real-time processing. Finally, *timing requirements* are the timing guarantees demanded of the network by any given station attempting to access the channel.

The real-time communication system must be efficient and reliable. It must be performance oriented in order to provide a very short response time while ensuring that network errors are detected and are recovered from without involving the network users. There is a trade off: as network error detection and recovery protocols provide more robust and hence more reliable service, the cost of executing these protocols on each protocol data unit increases, degrading performance. Thus it is desirable for a real-time communication system to implement only the necessary communication services. This is a minimum according to the required quality of service.

A major requirement of real-time applications is fault tolerance [HWAN87]. Generally, real-time applications produce specific network load conditions and traffic patterns; the load is relatively light and the traffic pattern varies slightly from predictable behavior [STOI88]. Errors within the application, however, must be detected, confined, and corrected using fault tolerance techniques built into the application. These fault tolerance techniques may cause unpredictable sudden bursts of high priority communication, are called *alarms*. An alarm message must not be stopped by flow control. This sudden peak in load due to alarms is called an *alarm avalanche*. Unless the communications subsystem can meet the deadlines of each of these alarms during an alarm avalanche the error condition will worsen, and more faults will be introduced. It is not sufficient for a real-time system to provide adequate performance during normal activity; the system must actively influence the ability of the fault tolerance techniques to confine and correct the faults before others compound. Real-time system designers must anticipate these message bursts, providing sufficiently robust transfer capacity for all network nodes under all load conditions. This requires high throughput and predictable message delays. Simply increasing the transmission bit rate cannot relieve congestion caused by burst traffic [MIRA83].

End-to-end resource allocation is also necessary to guarantee response times or throughput. Typically, this is accomplished with end-to-end flow control using windows and credits. However, this method only guarantees that buffer space will be available when messages arrive. Le Lann, in [LELA85], identifies at least two other types of resources that must be allocated, CPU cycles and I/O capacity. In fact, buffers have become a cheap and readily available resource, whereas most LANs are bound by the CPU and system busses.

Connections allocate resources upon creation. However, this allocation may be too costly to maintain if the connection is scarcely used or if there are many connections being supported. The robustness of a communications subsystem must not adversely effect the cost of maintaining the system, whether that cost is measured in resources or message delay.

The most notable and fundamental principle of a real-time communication system is that it must ensure that a message is delivered to its destination before the deadline. Its inability to do so is considered a failure. It may be both useless and dangerous to deliver a message past its deadline, as its significance is no longer a factor and its presence may confuse or overwrite data that did meet its deadline. The mechanisms used for meeting deadlines are priorities and multilevel message scheduling.

Latency control is seen as another requirement. Messages handled at all layers of protocols should be scheduled according to specific algorithms [LELA85]. Establishing finite upper bounds for access delay only solves part of the general problem. Finite upper bounds must be guaranteed for all service times at all layers for given conditions of utilization. Also, a cost function must be minimized when these bounds are exceeded. Such cost functions for a real-time system are the number of messages which miss their deadlines or the average or maximum message lateness over a given time interval. Deterministic message scheduling algorithms are needed to enforce the guarantee of finite upper bounds as well as ensure that the least costly messages are discarded on overload or in abnormal situations.

3. The Importance Abstraction

Importance is an abstract concept which ranks activities according to how much they contribute to the overall system goal. Within a system there are several subsystems, such as the communications subsystem. Within each subsystem are an arbitrary number of tasks requiring attention. These tasks are created by application processes within the system and are serviced by the appropriate subsystem. For the communications subsystem, these tasks are messages. A subsystem can be viewed as a server, where the application processes within the system are the clients which submit the tasks to the subsystem. In this section we present the importance abstraction, briefly as a characteristic of the general client/server model, then specifically as it applies to the communications subsystem. There are some aspects of the communications subsystem which make the importance abstraction more tractable; we will try to point out when the importance abstraction can not be generalized easily.

Informally, the system is doing the best it can to accomplish the system goal if each subsystem is also doing the best it can to service the tasks it has at hand according to which task will contribute the most toward the system goal. Somehow the subsystem must determine which task is the most important at each instant in time and service that task. The importance of a task is relative to the importance of all other tasks competing for the service within the subsystem, and the task with the greatest level of importance may change as the importance levels of the various tasks change. Therefore, for a subsystem, viewed as a server or a series of servers, to do the best it possibly can given a set of tasks and their relative importance levels, instantaneous preemption is necessary. Ideally, this preemption should save as much work already done as possible. Thus we consider a client/server model with the following characteristics: (1) each task has some function of time associated with it which describes the task’s importance over its lifetime and whose value is always known to the server; (2) the client must provide enough information to the server to determine the task’s importance over its lifetime; (3) at any point in time, all tasks are well-ordered according to their importance, and a *most important* task may be identified and served; and (4) the server is able to preempt tasks instantaneously without loss of work already accomplished. Thus a subsystem that can provide these characteristics will have the further characteristic that it will be doing the best that it is possible to do, according to what is globally considered important.

3.1. Importance Function Definition

Since the various subsystems are simply responsible for servicing the tasks submitted to them by the application processes, the subsystems must be given some mechanism for determining the order of the tasks to service. Within a client/server model, this is known as *scheduling*. Jensen *et al.* [JENS85] define a scheduling policy for operating systems by noting that within computer systems there is a value which varies with time associated with the completion of a process (an operating system's task). This value function, $V(t)$, is the value to the system for completing that process at time *t*. The goal of the scheduler is to optimize the cumulative value over the lifetime of the system, so that scheduling decisions are made based on what process can contribute the most value.

As a process's value function may increase and decrease with time, scheduling the current highest-valued process (greedy solution) often does not result in the optimal cumulative value. There are processes for which it is beneficial to "hold" them inactive until some time when the values associated with completing these processes have become optimal. This is not typically characteristic of a communications subsystem. Generally, no message is held once inside of the communications subsystem; it is delivered (completed) as soon as possible. Therefore, a communications subsystem which does the work at hand until no work is left will likely not produce an optimal value, since messages will not be held until their value functions have reached a maximum.

Rather than a function that provides the value of completing a message at some time, the application process should submit some function associated with the message which will describe the message's *importance value* over the lifetime of the message. This importance function is different from Jensen's value function in the following way. Instead of trying to optimize the *value upon completion* of a message, the communications subsystem is using the importance value at every moment in the lifetime of the message to rank that message among other messages, so that it can be working on the *most important* message at all times. The function which describes the message's importance is based on some set of system and environment parameters and maps these into any well-ordered set, for example the integers.

Since the system parameters are changing with time, every importance function, $I(t)$, is at least implicitly dependent on time.

There are variables implicit to the system which are measures of conditions which exist within the system. These variables trace the changes in the system according to outside influences or internal activities. An importance function may need to be responsive to these changes, since a message's importance may increase or decrease as these conditions on or within the system change. These system variables must be parameters to the importance functions, and are therefore called system parameters. However, the importance function may not use a particular parameter; in fact, we shall present importance functions that use none, some, and many system parameters.

3.2. Importance Function Issues

Now that we have defined a class of functions that maps a message's importance to the system into a well-ordered set, we raise some issues about this class. The design of the importance function controls the precision with which the importance function represents the value this message contributes to the overall system goal; high precision schemes generally entail more complexity. These schemes may be classified into *types* of importance functions. Furthermore, various applications have differing needs of a discrimination scheme; most do not need a complex scheme but a few require one. We offer a taxonomy of application's needs. Also, the more the importance function can describe, the greater the need to calibrate the function (relative to the number of "levels" possible) so that its values are comparable within the well-ordered set. This is called *homogenization*. Finally, it is observed that the importance function should be evaluated infinitely often to ensure an accurate ranking of messages. Since this is unrealistic, decision points must be assigned. The degree to which a communications subsystem can provide evaluation of the importance function infinitely often is called *granularity*.

3.2.1. Taxonomy of Types of Importance Functions

The taxonomy of possible types of importance functions is laid out in the tree of Figure 1. Generally, an importance function may actually change with time or may maintain the same value as time elapses. The former are called *dynamic* schemes, the latter, *static*. Static schemes are first broken into two cases: the very special case of one priority for all tasks (that is, no discriminating scheme at all) and multi-level static priorities, where $I(t) = n$ or $I(t) = 2^n$. The multi-level schemes are further divided into the trivial scheme with only two classes of tasks, a scheme with a small enough number of levels of importance that a human being could understand and keep the distinctions between levels in his head, a scheme where a human being

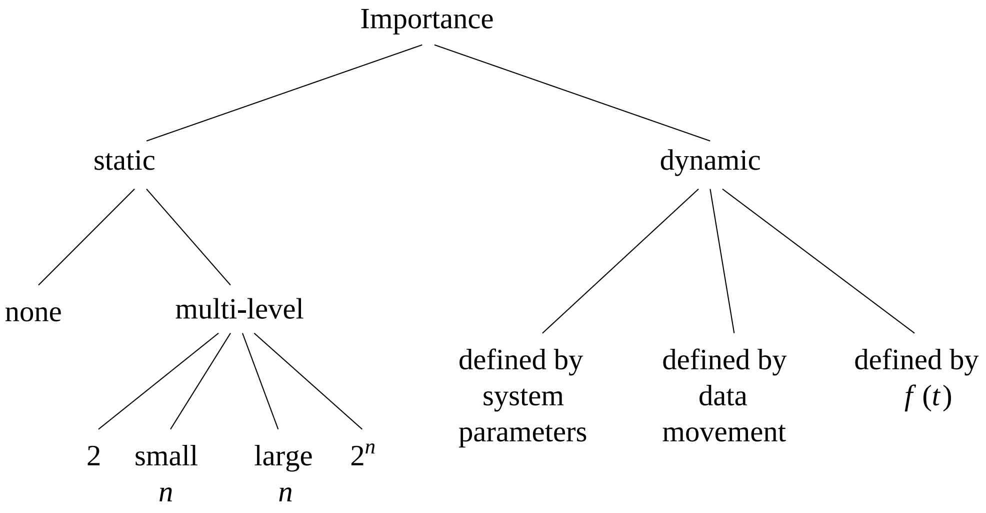

Figure 1 — Taxonomy of Types of Importance Functions

with the aid of a handbook could look up the distinctions between levels of importance but could not keep them in his head, and a scheme where the number of levels is so high that they can only be meaningfully assigned by an algorithm and not by a human being. Dynamic importance functions could be defined as either explicit functions of time or by some set of system parameters, in which case importance implicitly depends on time.

The most general importance function is one that depends on all of the system parameters. The system has many such parameters, like load, congestion at particular stations, level of criticality, and time. Each system parameter may itself be considered a function of time: load and congestion change over periods of time, criticality depends on what is happening at a certain time (such as battle conditions), and time certainly changes. Therefore, the importance function can be described on a planar graph with time as the dependent variable. Unfortunately, since there is no way of knowing what the system parameters will be in the future, there is little hope of graphing the importance function into the future. (It should be noted that time is a system parameter which can be predicted, so an importance function which depends solely on the current time (such as deadline scheduling) has the property that it may be graphed into the future.)

3.2.2. Taxonomy of Applications

Applications have differing needs for a discrimination scheme. It is desirable to be responsive to all types of applications. Figure 2 shows a pyramid of discrimination schemes according to application’s needs. At the top of the pyramid sits a scheme suitable for time dependent traffic, the *deadline traffic scheme*, which provides the most flexible and most complex method of assigning priorities. Only a tiny fraction of all applications, namely real-time applications, will need the full power offered by this scheme. A somewhat larger number of applications will desire a *k*-level priority space. This class of applications will be said to

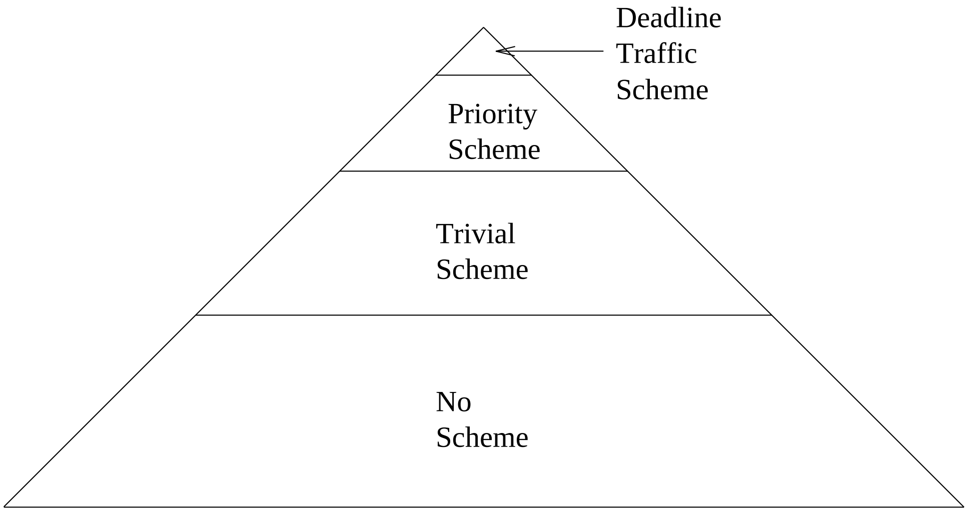

Figure 2 — The Pyramid of Discrimination Schemes

need simply a *priority scheme*. A still larger set of applications require only a trivial scheme, having *normal* and *extraordinary* tasks to perform. Finally, the broadest class of applications, the base of the pyramid, will not deem any discrimination scheme to be worth the effort and will have all its communications work done at the same priority.

3.2.3. Homogenization

Under every circumstance all of the messages in the subsystem must be well-ordered; that is, the ordering implied by the importance function must reflect the actual order of importance of the messages. Consider deadline scheduling where the message requiring delivery the soonest is the most important. Using only the system parameter of time, the set of messages in the system is well-ordered based solely on the difference between the time-to-serve and the current time. However, if there are also messages which have a static priority (a function of the *null* set of system parameters), making accurate decisions regarding which message is the most important becomes a matter of comparing deadlines against static priorities. In order for the importance function to produce a scalar value from the set of system parameters, each of the parameters must be weighted appropriately. In other words, there must be some *homogenizer* which can ensure that all messages, regardless on what set of system parameters they depend, are well-ordered according to which messages can contribute most to the system goal. The question remains: how do messages with different importance functions, and hence different criteria for importance, relate to each other?

One method of trying to simplify the problem is to enumerate all of the clearly useful importance functions. Now the problem becomes one of relating a finite number of functions at the expense of complete generality. The homogenization problem is simplified further by restricting the system to a small, closed environment. Designers of a closed system may have some idea of the characteristics of the system and can adjust the importance functions accordingly. At this time, this is considered too restrictive, since wide area networks with many gateways and high bandwidth are soon to be a cornerstone of the scientific computing community.

3.2.4. Granularity and Decision Points

For the statement "the communications subsystem is servicing the most important message at all times" to be true, the subsystem must be able to know the values of importance for each of the messages in the system at each point in time. Since time is continuous, the importance function must be evaluated infinitely often so that at the instant when a message’s importance value supercedes the current most important message, that message may be preempted and the new most important message may be served. In reality, this is impractical.

The degree to which a communications subsystem can adhere to the above statement is called *granularity*. As the granularity becomes more gross, the communications subsystem can ensure the most important message is being serviced at *almost* every point in time.

Thus, the importance abstraction provides a method for considering a system which has the basic principle that it is always trying to do the most important activity according to what contributes most toward the system goal. Unfortunately, the characteristics of the pure importance abstraction are impractical; no server can preempt messages instantaneously without loss of work, nor can an importance function be evaluated infinitely often at no cost for computation. The importance abstraction is the right place to start, however, as long as relaxing the assumptions mentioned provides a close approximation to the pure scheme. For example, the importance functions may be evaluated periodically so that during the period no changes occur in the ordering of messages. This period should be chosen so that the calculated importance values could be updated faster than the actual importance can change substantially. This period may be no smaller than a bit-time and still be meaningful, since a bit is an atomic entity within computers. Another scheme is to define decision points based on the observation that there are certain times in a message's lifetime when it is possible to reorder the messages, so that a new most important message may take the place of the message that used to be the most important. These decision points include at least message entrance into the communications subsystem, transmission onto the medium, servicing within gateways, and exiting the communications subsystem.

4. Applying a Discrimination Policy to a Communications Subsystem

The goal of this research is to find a scheme that will provide a discrimination policy that is flexible and potent. It needs to be flexible enough to apply to all types of networks, from coast-to-coast wide area networks to tightly closed control networks. It must apply to all types of traffic as well, from real-time to background communications. It must be consistent at every installation, yet make no assumptions about the environment in which the network resides. In order to suggest a reasonable scheme to be used within a communications subsystem, we must first observe what most classic transport protocols cannot do with respect to the abstract notion of importance. We can then observe what the transport protocol can and should do. We also recognize a set of open questions and discuss their implications.

4.1. What a Communications Subsystem Cannot Know

A communications subsystem built upon a classical transport layer protocol is not endowed with any special characteristics that would make it suitable for a real-time system. It provides the reliable, end-to-end data transfer service to its users as a part of a communications subsystem. However, there are aspects of the system that the transport protocol cannot know yet ignorance of them will hinder its responsiveness to real-time needs.

(1) Obviously, no protocol can control for bit errors on the medium or packets lost in the network. These errors destroy any latency guarantees that are not probabilistic.

(2) The "system goal", as defined by the system designers, is neither the communications subsystem’s business nor responsibility to understand. As part of the communications subsystem, the transport protocol must provide the best service it can, based on the requests issued by the application processes.

(3) Likewise, the communications subsystem cannot know what the enclosing environment is, such as whether the area of distribution is wide or local. There are parameters within the transport protocol which can be tuned and adjusted for its various uses, but the protocol itself should remain consistent under any environment. Specifically, the discrimination mechanism must rely on the same principles under any environment as well.

(4) The communications subsystem cannot assign values of importance to the messages it handles. These messages are important only to the application processes which create or use them, and a message's system-wide importance only makes sense to entities which know the system goal. It is the responsibility of the communications subsystem, and the transport protocol specifically, to be responsive to the needs of the application processes by enforcing the discrimination required; it is not the communications subsystem's responsibility to make up the discriminators it must enforce.

(5) Typical transport protocols are designed to work in conjunction with any physical (and MAC) layer protocols which provide the common basic frame transfer services. It is not known to the transport what protocols are providing these services, so it is not known how to cause the lower layers to be responsive to the discrimination scheme that it is trying to enforce. For instance, Ethernet and FDDI are both considered viable lower layer protocols for use with various transport protocols; however, Ethernet has no medium access priorities and FDDI has infinitely many of them. Yet, the transport protocol cannot know with which protocol it is being used. Furthermore, it cannot know the signaling speed or the propagation delay between stations, or how many routers and/or bridges there are on the network.

4.2. What a Communications Subsystem Can and Should Know

In spite of things that a communications subsystem cannot do, there are some things that it can and should do to provide some form of discrimination among messages according to their importance to the system. The communications subsystem controls several resources for which messages may contend. These resources include buffer space, processing attention, and network access. The most important message should not have to wait on a resource, since that would imply that a less important message is preventing access to that resource. This is called *priority inversion*. The subsystem should periodically order activities according to which activity has the highest importance value. At each decision point it should give preferential treatment to the most important activity, where a decision point is an opportunity for reordering activities.

With any scheme for providing discrimination, the hope is that some class of messages will benefit from the shedding of the load of all classes of less importance. A message with moderate importance may be impeded by messages of more importance; likewise this message may impede the progress of messages of less importance than itself. Degraded service of less important messages due to this impedance is the price for providing preferential treatment of the more important messages. The most important message should be the least impeded. Furthermore, impedance on the most important message should be only slightly more than the impedance of a message on a zero-loaded network without any discrimination scheme at all.

Proper management of system resources within the realm of a communications subsystem means avoiding priority inversion and reducing the impedance of the more important messages. One way to ensure that the subsystem has the power to accomplish these objectives is to allow the preemption of less important tasks in favor of more important ones. Preemption is a powerful and costly tool: powerful in that the communications subsystem can take away a resource *at any time* in order to give it to a more important task than the current one, but costly in that when a task is preempted either work will be lost or there will be a good deal of overhead in saving the state of the work. Hence the use of preemption must be sparing else the performance penalty will cancel all advantage.

However the discrimination scheme is implemented within the communications subsystem, a doctrine concerning such schemes must still hold: there should be varying degrees of preciseness, and the price for such a scheme should increase only with the increased precision. As the scheme becomes more precise about which message is the most important, the cost of determining this message will increase. Having no scheme is to have a scheme that does nothing; this scheme provides no discrimination among messages but should provide the highest overall system throughput, since no load of messages are being shed. There should be no overhead due to a discrimination scheme if one does not use the discrimination scheme. Four tiers of precision are useful: none, trivial, static, and precise. No scheme provides only a normal data transfer service, it is up to the application process to discriminate. A trivial scheme is a binary division, perhaps like ISO Transport Protocol Class 4's normal and expedited services. The static scheme provides a range of priorities, which, once set, are not changeable. Finally, the precise scheme provides a dynamic priority which is raised and lowered by the measure of importance that message has to the system. These four schemes should be able to coexist. Clearly, the more precise the scheme, the more it will cost to calculate which message is the most important. Herein lies a delicate trade-off: if the ability to provide a very precise method of preferring one message over a set of others costs more than the benefit gained by the preferred message, the scheme only hinders the system performance. A discrimination scheme is an investment which must show tangible benefits to the class of messages whose performance it wishes to improve.

As suggested in Figure 3, no discrimination scheme at all results in a first come first serve (FCFS) service discipline. Any attempt to order messages into importance classes is equivalent to having multiple queues which the server must drain. Since having no scheme requires less effort, it must be that the throughput for the FCFS system is higher than the system with discrimination. As a doctor's first rule is to do no harm, so a protocol designer must be sure that any proposed discrimination scheme does not result, due to the computational overhead of determining the importance of each message, in a lower latency for the class of preferred messages than what would have resulted from an unintelligent FCFS discipline.

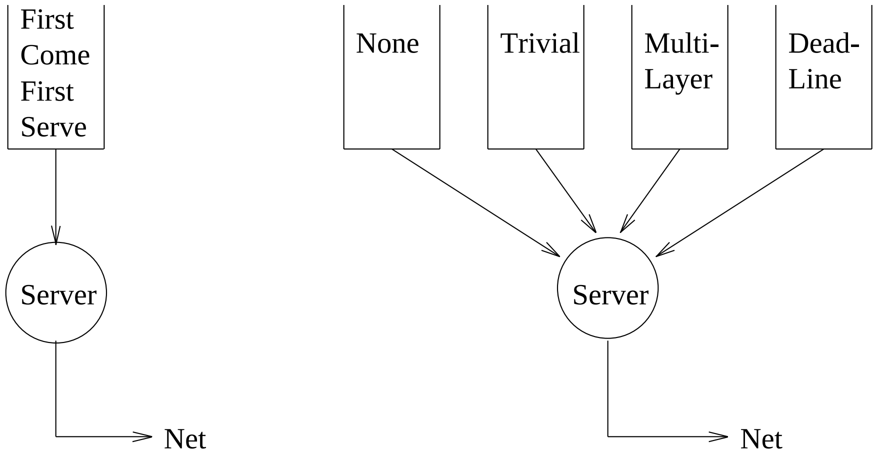

Figure 3 — Unintelligent *vs.* Discriminant Schemes

The communications subsystem should provide a good interface to the user, regardless of how precise the discrimination scheme may be. The interface should be simple, low overhead, consistent, orthogonal, regular, and clear. The scheme should be able to operate unambiguously from the parameters which are provided to it by the application process.

Regardless of the scheme used, a message placed into the communications subsystem should be delivered as soon as possible, according to its relative importance with the rest of the messages contending for the attention of the subsystem.

4.3. Open Questions

There are many questions which must be answered before the definition of a priority field can be made completely.

(1) What are all of the decision points in a communications subsystem, where activities may be reordered if warranted by the discrimination scheme? Since a transport layer protocol cannot control the layers below it, its only realm of influence is the processing it performs on the message itself.

(2) The interface to the communications subsystem must convey enough information to allow the subsystem to be responsive to the relative importance values of the messages. What parameters must be included in this interface? How will the scheme use them? Also, how will the transport layer protocol, once it knows a message's importance, convey that importance to the underlying services, without knowing what they are?

(3) Real-time applications are by definition constrained by time. How necessary is it to have a time-based discrimination scheme? If it is necessary, common time must be distributed to all network nodes (not an easy task), and it must be an order of magnitude more accurate than is necessary. Algorithms for time distribution exist ([GORA]), but they are complicated and hinder efficiency. The communications subsystem cannot get the system time through system calls; this is surely too costly. Perhaps the subsystem should have its own time of day clock. Even so, accuracy is a major issue, and a time-based scheme would be gross at best. Real-time application processes may have deadlines and time constraints, but it is not clear that the communications subsystem really can enforce deadlines. It can give preferential treatment according to a deadline and some other weighting factors, but the deadline time itself may not actually need to be included in the priority field.

For the purposes of this discussion a deadline is a time by which the receiving operating system must be notified that a message has arrived. This definition is reasonable since the communications subsystem can do no more than deliver a message to an exit queue. After that point the message leaves the communications subsystem’s sphere of influence.

Even assuming that having an accurate system clock is a solved problem, the use of deadlines remains problematic. A deadline handed to the communications subsystem by an application process is not meaningful for calculating importance. This raw deadline contains no information about the time that it will take the message to be delivered and hence does not accurately reflect the urgency of the message’s delivery from the communications subsystem’s point of view. To make a deadline, *D*, useful in ascertaining importance, the following times must be known (or estimated): time required for the subsystem to seize the necessary resources (t_{SR}), time to gain network access (t_{NA}), time to transmit (t_{TR}), propagation time (t_P), time in routers and gateways (t_{RG}), and time in receivers (t_{RVR}). If these times are subtracted from the deadline, one gets the "latest time to transmit", t_{TRANS}, which represents a "true deadline" from the communications subsystem’s perspective. A plausible discrimination scheme then would be to service messages by an earliest transmit time first discipline:

$$t_{TRANS} = D - t_{SR} - t_{NA} - t_{TR} - t_P - t_{RG} - t_{RVR} \tag{4.1}$$

Unfortunately, many of the variables in Equation 4.1 are unknown, are hard to measure, or have a very high variance. There is no way to know or predict with accuracy how long a message might spend in routers and gateways or in an entrance queue waiting for access to the network. The only time that is easily computed is the time to transmit which is simply the size of a message divided by the capacity of the network. But this fact is of little use since the t_{TR} variable is one of the least significant terms in Equation 4.1. Moreover, even if the intractable problem of computing t_{TRANS} were solved, an underlying assumption in earliest transmit time first is that two messages with the same deadline are equally important, which may not be the case.

(4) It is not clear that a message will be in a communications subsystem long enough for its importance value to change. This surely depends on the environment; wide area networks with several gateways may impose a large latency on messages. If all messages can be delivered before any appreciable change occurs in their importance values, then a dynamic scheme may not be necessary. It is probably true that some scheme is necessary to resolve contention, whether this is dynamic is an open question.

(5) What assertions and guarantees fall apart during the presence of errors in the communications subsystem? Transport layer protocols provide a reliable service, which implies some need for acknowledgements or negative acknowledgements. Latency may be as much as two orders of magnitude larger in the presences of errors than during error-free operation. Resources remain tied up while the transport layer tries to recover. Under these circumstances, is it clear that a static scheme is all that is necessary? What importance values should acknowledgements and retransmitted data carry?

These are a few of the observations and questions our research has lead us to discover.

5. Toward A Workable Discrimination Policy

In general, each message in a subsystem must have some method of encoding the importance function within it so that the communications subsystem can choose the most important message to serve. As the system parameters change, the value of the importance function changes to respond to the new environment. The communications subsystem can evaluate the importance function as it needs to know a message’s importance value; ideally this evaluation would be done infinitely often with no cost so that the instant one message becomes the most important message over the previously most important message, the communications subsystem can preempt the old message and serve the new one.

Admittedly, this scheme is unrealistic for any subsystem, particularly the communications subsystem. It is nonetheless the place to start since the assumptions which are made about the communications subsystem can be relaxed later. This scenario represents the best that can happen because the system is in essence being driven by an oracle which can properly assign importance functions to each message and thus determine how the communications subsystem can be most responsive to the system's overall goal.

First and foremost among the difficulties with the abstraction of importance is how the applications can assign importance functions that will accurately describe a message's importance during its lifetime. In order to do this the system designer must know *a priori* what other messages are possible at every moment during the message's lifetime, what their importance functions may be, and furthermore must ensure that some message does not preempt a message which is really more important or is not preempted by a message which is really less important due to an improperly assigned importance function.

In this section we examine some aspects of the importance functions which produce some "useful curves". These curves represent the mapping of an importance function onto a planar graph with time as the abscissa. An importance function may be based on many aspects of the system, we show graphs with only one aspect contributing. We also discuss the problem of mapping many individual importance functions into a common planar space. This mapping is called *homogenization*. We conclude this section with some ideas on implementing a workable discrimination policy, complete with tradeoffs and compromises, in two subsections describing example schemes.

5.1. Message Types and Useful Curves

While it is not clear that a canonical list of message *types* exists, one possibility for the communications subsystem includes: deadline, dynamic priority, static priority, and no priority scheme at all. Each message's importance is derived from what is important to it. For example, if it is important for a message to be delivered before a deadline, the importance function will depend upon the system parameter of time and the message's deadline. Other parameters, such as whether the message has a hard or soft deadline, will help rank it among other deadline traffic, and in fact, among all other traffic.

Although the importance abstraction is powerful enough to provide a system with the mechanism necessary to impose a ranking among messages competing for the communications subsystem, without a canonical list of the message types it is virtually impossible to write down a function which will adequately describe a message's importance over its lifetime. The value of an importance function $I(t)$ changes as the set of system parameters upon which it is based changes. How the value changes depends on the nature of the importance function. There are many possibilities to consider: should $I(t)$ be linear, stepwise, or exponential? Is $I(t)$ decreasing, increasing, or monotonic? Is $I(t)$ everywhere defined? Based on our own experience, experience reported in the literature, and discussions with several system designers, we have attempted to draw several *shapes* of curves that might be useful importance function shapes.

The simplest to draw and to understand is the static priority scheme, Figure 4, where the application process provides the message with a number, or fixed importance level, which it will keep during its entire lifetime. The origin of the graph, t_0, is the birth of the message, or when the message enters the subsystem. Figure 5 shows a hard deadline example where the importance increases exponentially to infinity as the critical time approaches. If the message is

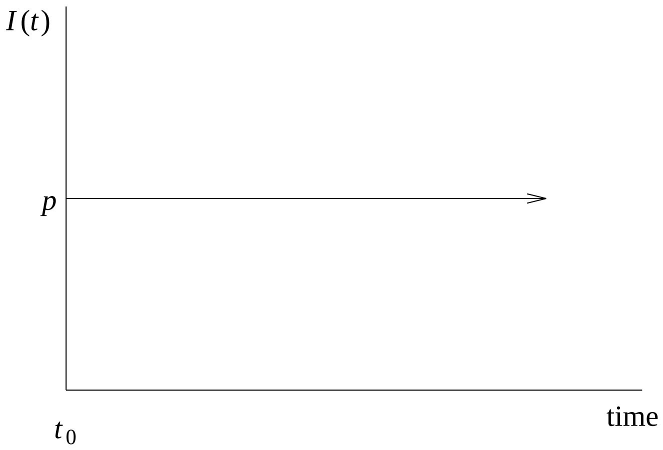

Figure 4 — Static Priority

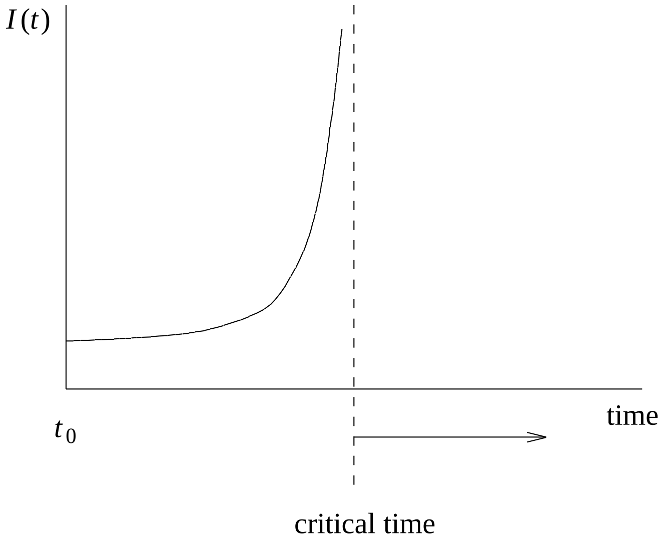

Figure 5 — Hard Deadline

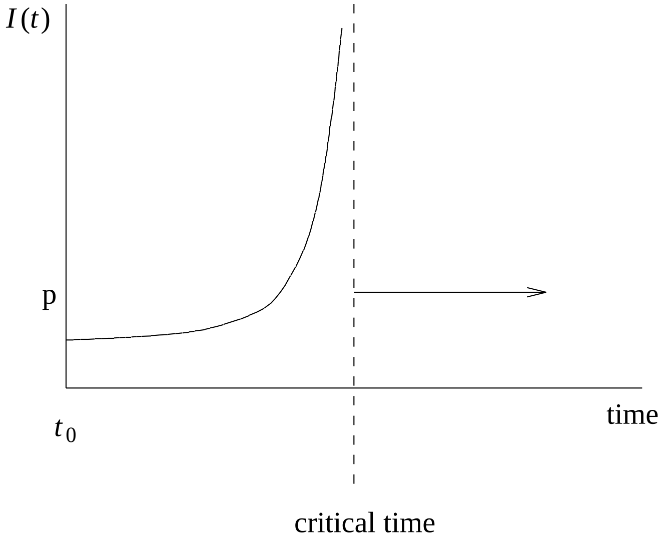

Figure 6 — Soft Deadline

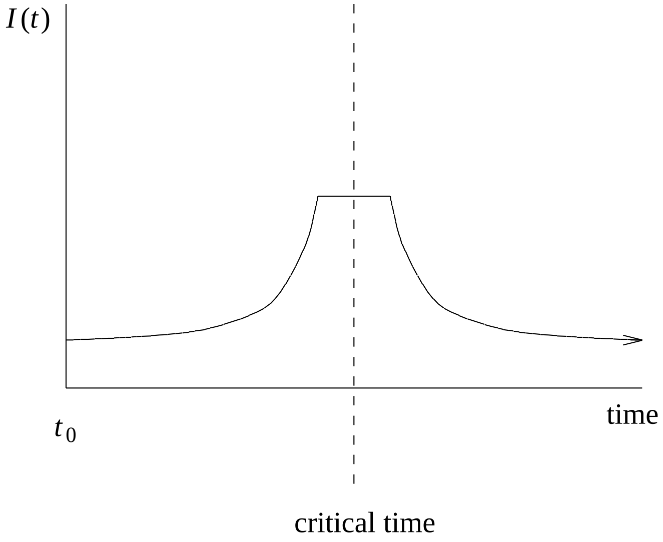

Figure 7 — Window of Opportunity

not satisfied by the critical time, its importance value becomes negative, indicating that it is very *un*important. Figure 6 shows the same importance function before the critical time is reached, only now the message retains some importance even if the critical time passes, as might a message with a soft deadline. Figure 7 shows an importance function of a message that has a "window of opportunity", where the message's greatest importance value occurs over an interval of time centered on the critical time. Notice that in this case the importance function is symmetrical about the critical time. This indicates that the message is as important to satisfy some time before the critical time as it is to satisfy the same amount of time after the critical time. Of course, there are many variations on these themes; the point was to show some useful shapes of curves which may be used for the importance functions of certain types of messages.

5.2. Ideas on Homogenization

Another problem with the importance abstraction is that the values of the importance function must somehow relate to one another in order to impose a ranking. Thus, if one importance value is higher than another, it is defined to be more important. However, in order to ensure that all importance functions, and there can be infinitely many of them, are mapped onto a planar space in a manner which is consistent, each importance function would have to be carefully calibrated. Mapping all possible importance functions onto a common planar space is called *homogenization*. For example, consider an importance function of one message that is based on a deadline such that the importance of the message is exponentially increasing as the deadline approaches, such as Figure 5. Also consider the importance function of another message which is static, like Figure 4. How are these messages related? At what point should the deadline message become more important than the message with static importance? What parameters must be included so that the two functions share the same planar space? Each possible message's importance function would have to be compared to each other's importance function, which is combinatorially explosive.

The following is an idea which can simplify matters without compromising the integrity of the importance scheme. It is not clear at this time whether the premise is completely correct. The premise is: all messages which are dependent on deadlines are *inherently more important* than any other messages. Thus, all messages can be split into the two categories: deadline and priority. Priority messages are messages whose importance function is either a constant (static priority) or implicitly dependent on time by being explicitly dependent on some system parameters which are dependent on time (dynamic priority). Deadline messages no longer have to share the same importance space as priority messages; there is now an importance space for each category of messages.

Our reasoning which led to the above premise was that the communications subsystem cannot appreciate the consequences of failing to meet a message’s deadline, therefore the communications subsystem relies on the application to assign deadline driven priority only to those messages for which the consequences are profound. No other message can be as inherently important as a message with deadline driven priority since it has no concrete point of consequence.

In terms of implementation there could be two queues, one that dealt exclusively with deadline messages, and one that dealt with all other messages. Within these queues, messages would be ordered by their individual importance functions; however, all messages in the deadline queue are satisfied before any message in the other queue is. This clears up the question about how deadline and non-deadline messages are related — the deadline messages are *always* more important.

It should be noted that there is at least one important exception. An alarm message is always more important than any other message, except possibly another alarm message.

Alarms do not necessarily have deadlines, but would be more important than any deadline message. To handle an alarm by the scheme above, one would have to artificially treat it as a message with a deadline, even though no deadline is appropriate.

5.3. Example Scheme Number 1

Another idea has more promise. It is based on the observations that (1) calculation of an arbitrary function to determine the importance of a message may be too expensive, and thus actually hinder rather than help the application processes; (2) the importance function should be evaluated fairly often to catch any changes; and (3) a series of static importance values may come close enough to approximating a continuous importance function to be usable. The following discussion builds on these observations.

The priority field associated with a message will contain an encoding for three decision times, call them t_1, t_2, t_3, at which the importance may be raised or lowered. (Note the existence of decision times is predicated on a common time reference within the communications subsystem.) Also encoded for each region on the time line will be an integer in the range -1 to some maximum, *n*, where higher numbers are assumed to represent higher importance values. The priority values for the four regions of the time line, that is p_1, p_2, p_3 and p_4, may be assigned any value from -1 to *n*. Figure 8 shows how the time line is divided.

Recall the four classes of priority schemes shown in Figure 2. All can be accommodated in one importance space under our proposed mechanism. Time dependent traffic, or *deadline traffic,* is inherently more important than traffic that is not time dependent. For this reason *deadline traffic* will have the use of all possible priority values, -1 to n. *Priority traffic* will have the values from -1 to $n/2$ available to its importance functions.

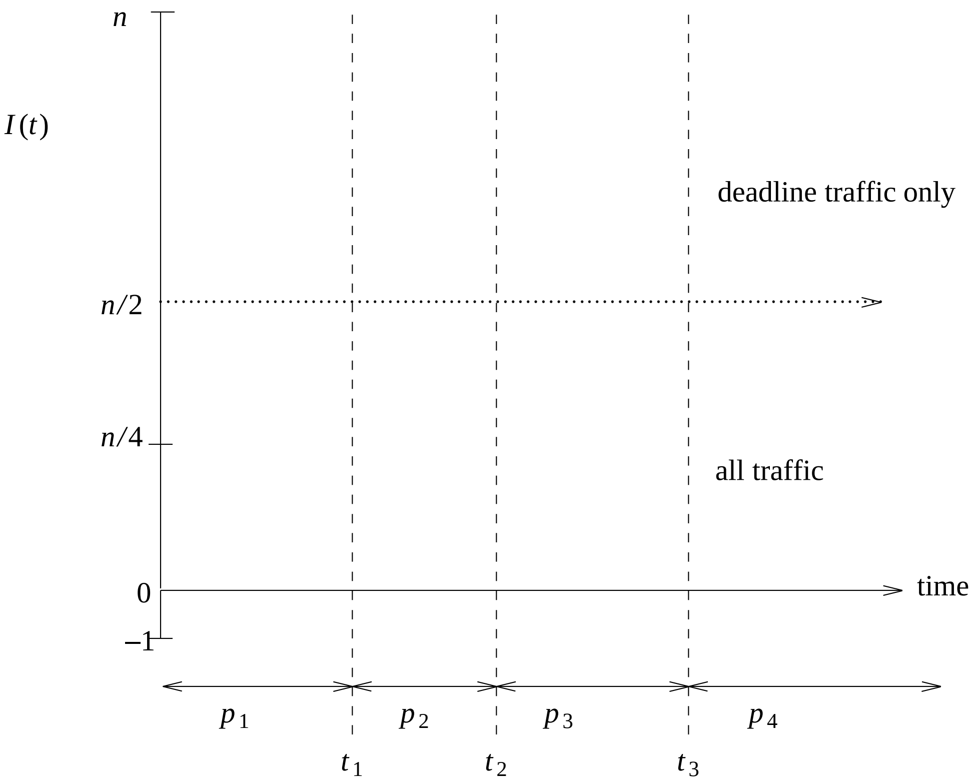

Figure 8 — The Importance Space with Three Decision Times

The value −1 will represent the *kill* priority, which provides a mechanism for controlling the lifetime of a message within the communications subsystem. The value of the importance function can be computed by determining into which region of the time line the current time falls and reading the priority value for that region. If the value is −1, the message is immediately discarded.

To illustrate, consider Figure 9 where the hard deadline case of Figure 5 is modeled. The deadline (or critical time) can be encoded as time t_3 in the priority field. Initially the importance of the message is small, so the priority may be chosen to be less than $n/2$ to allow the system to service other (possibly non-deadline) messages. At time t_1 the importance jumps to a higher priority. In particular it jumps to a priority above $n/2$ so that this deadline message is now assured of preempting all traffic in the lower priority schemes. At time t_2 the importance goes still higher since the approaching deadline increases the urgency of performing the message. Finally, since the semantics of a hard deadline application dictate that the message should not be delivered if it misses its deadline, after time t_3 the message is given the kill priority. (To model instead the soft deadline of Figure 6, p_4 would be some (presumably low) non-negative priority.)

An important aspect of this scheme is its ability to ensure that during certain time intervals a deadline message does not have to compete with any non-deadline traffic. A deadline message is assumed to be inherently more important as the deadline approaches than any time independent traffic; hence the highest priority values are reserved for use by deadline traffic only. On the other hand the overall efficiency of the communications subsystem and its fairness are improved by allowing time dependent traffic to be assigned priorities less than $n/2$. A deadline message's importance is indeed low when the deadline is far into the future or, in some cases, after the (soft) deadline has passed.

If desired, the deadline can be encoded as time t_2. Now, when the deadline is passed, a message may have its priority reduced in the interval between time t_2 and t_3 and then be killed after time t_3. In this way a message that has only an interval of time after its deadline during which its delivery would be useful can be accommodated.

An exact deadline means that the message will be most effective if delivered precisely at the deadline. This is the "window of opportunity" illustrated in Figure 7. Here the importance function should have p_2 and p_3 as the same (high) priority and p_1 and p_4 as the same (low) priority. In this way the message is most likely to exit the communications subsystem in the

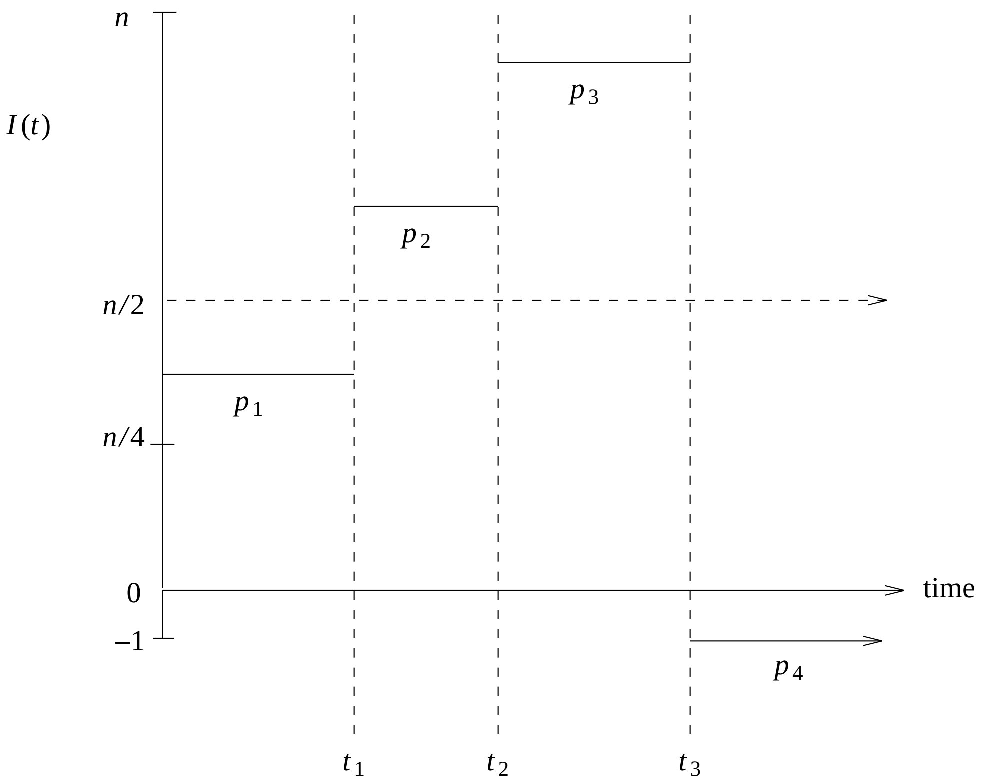

Figure 9 — An Example

interval from time t_1 to time t_3.

Some applications will need only a *k*-level priority scheme for their traffic. For dynamic priorities the importance value of a message may be raised or lowered at each of the decision times t_1, t_2, and t_3. An application achieves static priorities by assigning p_1, p_2, p_3 and p_4 to all have the same value. A common use of p_4 will be to control the lifetime of a message; that is, p_4 may be given the *kill* priority as its value so that a message will not remain in the communications subsystem beyond a certain time, t_3.

Applications requiring only a trivial priority space, that is, one with two priority values, or no scheme at all, need only static priorities. For the trivial scheme *normal* traffic should be mapped to priority 0 and *extraordinary* traffic to some middle value in the importance space of all *priority scheme* traffic, namely $n/4$. If it uses no scheme at all, the application has indicated no concern about discriminating between messages, and the communications subsystem cannot divine importance on its own. The lowest importance value, 0, is thus the logical value to assign all messages from these applications.

5.4. Example Scheme Number 2

Recalling the principles we set forth in Section 4, we present another flexible and potent scheme for use within a transport layer. The following is a description of the syntax and the semantics of this scheme, and how a transport layer would provide different *classes* of service. This scheme is examined for its ability to provide an application process (i.e., the transport user) with the ability to assign its messages an importance according to the type of service they would need.

A *cleanpoint* is defined [CHES88b] as a point in the sequence of operations where data structures are in a stable and consistent state and can be modified without danger of race conditions or inconsistent behavior. It is expected that the maximum time to a cleanpoint can be bounded in terms of packet transmission times. As a packet is being transmitted, another, more important one, may become active. Once the first packet is completed, a cleanpoint exists for the choosing of the next packet to transmit; the most important one of those pending is chosen. A cleanpoint must exist at each scheduling point so that the activities at that scheduling point may be ordered.

There are at least four places throughout the communications subsystem where urgency values should affect communications: entry scheduling, media scheduling, gateway scheduling, and exit scheduling [CHES88b]. The output message selection is the entry scheduling; associating *sort* values with priority queues allows the user to manipulate the entry scheduling. Media and gateway scheduling may be seen as an extension of entry scheduling. Exit scheduling is the order in which messages are passed to the application process.

Consider an *importance* field which is 16 bits wide for use in transport packets. Of those 16 bits, the first 4 are reserved bits and the last 12 are used to provide the mechanism for discrimination among transport protocol data units (TPDU packets). The semantics of the *importance* field are supported within the transport protocol as the discrimination policy.

Each packet carries the connection’s *importance* value in its *importance* field, including data packets as well as control packets. This implies that acknowledgements are as "important" as the packets they are acknowledging. Thus a connection is said to have an *importance* value, which implies that all packets generated by that connection will likewise have that *importance* value. The user of this connection may change the importance value of that connection at any time during the connection’s life, giving a dynamic nature to its importance and allowing the connection to respond to the (possibly) dynamic nature of the user.

There are three *classes* of priorities: normal, expedited and preemptive. Normal packets adhere to all flow and rate control and local allocation policies set up for that context and enforced by the transport layer. Expedited packets are more important than normal packets, and at every cleanpoint point all contexts with the expedited *importance* value are served before any context with a normal *importance* value may be served. The expedited packets are allowed to bypass the normal flow control restrictions. Preemptive packets are the most important, and all contexts with the preemptive *importance* value are served before any other context is served.

These packets are allowed to preempt any other packets in progress to ensure instantaneous use of any transport level resources it may need. These packets move through the system as if no other traffic were there to impede their flight. Therefore, a preemptive packet may seize buffer space if none is readily available, destroying what had previously occupied that buffer space, which is the tradeoff for very low latency communications.

The priority space provided by 12 bits is 2^{12}, or 4096 levels. Within this space, the highest level is reserved for the preemptive class: level $2^{12}-1$ (all ones). The second highest level is reserved for the expedited class: level $2^{12}-2$. All other levels are left for use within the normal class: levels 0 through $2^{12}-3$. Connections with higher *importance* values will be served before connections with lower *importance* values. At each decision point, once the packet is in flight, the packet with the highest importance value will be served before any other packet. The two special *importance* values will be served according to the semantics of the class for which they are reserved. Note that preemption can only occur on those resources controlled by the transport layer; standard, non-preemptive services below transport are allowable without severely reducing the power of the expedited class.

The preemptive class fulfills the goal of reducing the impedance of a critical message to nearly the impedance on a zero-loaded network. A packet with the highest priority traverses the network with no resistance since it commandeers resources at every point of contention and waits on no other message. The use of preemption is potentially costly in the amount of work lost, but in a real-time environment the need for the lowest possible latency for critical (alarm) messages outweighs concerns about reduced throughput for less important messages.

Expedited messages will move quickly through the communications subsystem without suffering the constraints of flow control and without waiting on any normal traffic, yet expedited messages will not penalize the throughput of the system nearly as severely as preemptive messages might. This class of importance clearly provides a lower latency for its messages than that of normal traffic. That is, the investment of reduced throughput in using expedited messages does pay off in the form of reduced latency for this class of message.

The *importance* field is present in every packet, so the scheme applies consistently across the entire route of the packet. Every packet, data or control, is treated according to its *importance* value at every scheduling point throughout the route.

The semantics of the *importance* field are known to each transport implementation. In some protocols it was up to the end systems to agree on the meaning of the an importance field, which would allow various installations to define their own ranking schemes. That would be highly flexible but would lead to inconsistent uses of the *importance* field. This scheme retains flexibility while allowing the transport layer to enforce the meaning of the *importance* value in a consistent manner. The added benefit is that all packets traversing a gateway may be treated consistently without the gateway needing to know every installation's (previously agreed upon) interpretation of the *importance* field.

Furthermore, allowing the transport layer to enforce the discrimination scheme implies that the layer can become an active participant in providing classes of service to its clients. Some protocols are only allowed to rank the packets according to the *importance* values in ascending order. A packet with a low *importance* value could be caught behind a packet with a higher one until that packet's transmission was complete. That is priority inversion. Since there may be an arbitrary number of gateways between two end systems, a packet may suffer an arbitrary number of delays due to priority inversion. Because the proposed discrimination scheme allows the transport layer to aid packets that are very important, it actively contributes to the scheme. Expedited packets are allowed to bypass flow control; preemptive packets are allowed to stop the current transmission fatally. Although expensive, by enforcing these classes the transport layer takes on an active role in the latency control of packets.

The *importance* value applies to the entire connection. Since the connection is typically the conduit for a specific communications profile, the *importance* value applies to all packets generated by that connection. Emergency and fault isolation and recovery communications are typically independent activities, so they would be given their own connections. The connection of an emergency communications channel may have the preemptive class *importance* value; it may not be used very often, but when it is, it is very important that the latency is low. Bulk file transfer may have a normal class *importance* value; transmission is often but timeliness is not essential.

Because the *importance* value applies to the entire connection, it may be said that the message, which is made up of several packets, has the *importance* value of the connection. Also, control packets which are acknowledgements inherit the *importance* value of the data which they acknowledge. If data has a certain importance associated with it, the acknowledgement of that data must certainly be as important since the acknowledgement "closes" the resources outstanding due to that packet.

The normal class packets obey all of the flow and rate control and local allocation policies within the transport layer. Expedited class packets are allowed to bypass the flow control. Flow control prevents a transmitter from creating congestion and overruns from lack of buffer space. Rate control differs in that it is how fast a node can handle inbound packets, not how fast the buffer space becomes available but how fast the transport implementation can handle the inbound packets. Expedited class packets should retain the rate control as it would be foolish to pump packets into a receiver that was not physically capable of keeping up. Bypassing flow control may mean that some buffer space is overrun, but this is the price for guaranteeing that packets will not have to wait on buffer space occupied by less important packets. Preemptive class packets not only bypass flow control but can fatally preempt a less important packet while it is being transmitted. The guarantee is that preemptive class packets have the minimum latency; the price is lowered throughput due to bumping less important packets.

This discrimination scheme also supports the notion of *criticality*. Expedited class packets may shut off normal class packets by taking precedence at each cleanpoint and by overwriting the buffer space occupied by a normal class packet if necessary. If the load of expedited class packets were sufficiently high, the normal class would essentially be shut off. Likewise, the preemptive class can shut off both the normal class and the expedited class.

There are some drawbacks to this scheme, however, that should be mentioned. The most notable is that there is no concept of time in the *importance* field. The *importance* field size could be extended to allowed an application process to define a large number of the bits to be an absolute time. This time could be used as a deadline in real-time systems. The 12 bits proposed here are far too few for a timestamp of any usable kind. It is unfortunate that time is not present, but a much more serious problem must be overcome before time's absence would even be an issue. Furthermore, bringing in the concept of time requires different *importance* values for all packets, even of the same connection. Current transport layers do not have a notion of global time, as discussed in the open questions of Section 4.3. Before scheduling with nanosecond accuracy is possible, all participants in the scheduling must be on common time to at least that granularity. As seen in the definition of the *sort* field in XTP Revision 3.4 (Section 6.1.5 of this document), static priority and clock time are overloaded into the same field, though they are never used at the same time within the discrimination policy.

It is not clear that only one level for the preemptive class, or only one level for the expedited class, or $2^{12}-2$ levels for the normal class is sufficient or necessary. In the normal class, 4094 levels may actually be excessive. The major contribution here is the division of service into classes for varying degree of latency control, not the number of levels within each class.

In effect this scheme falls under the heading of multi-level static scheme with the number of classes large enough that an algorithm must be used to assign importance values. The mechanism has all the advantages of static schemes, namely the avoidance of the difficult problems of calculating importance functions and comparing (possibly disparate) importance values. It does have the distinct advantage over a flat static priority space, however. The classes of packets allows a transport layer to be a full participant in the control of latency, which is essential in time constrained and other special needs communication.

6. Xpress Transfer Protocol

One of the legitimate criticisms of ISO protocols is that they were not originally designed for real-time applications; they were designed for interoperability, not performance. In a test of protocol performance over an Ethernet at Silicon Graphics Inc., it was reported in [CHES87a,b] that, of the 10 Mbps signaling speed on an Ethernet, 6.7 Mbps were available at the MAC layer, 4.5 Mbps after passing through the internet layer, 2.8 Mbps at the top of TCP, and 1.2 Mbps were available to the application program. Some measurements of ISO protocols, as in [STRA88a,b,c], reported even lower throughput. Two other problems often encountered are: (1) In TCP and ISO Transport Class 4, flow control operates end-to-end, so gateways do not regulate traffic. This increases the probability of congestion in the gateway, which in turn increases the probability that messages will be lost and retransmitted, further reducing performance. (2) Internet gateways add buffering to the connection, thus increasing end-to-end message latency.

These observations inspired the Protocol Engine Project, whose goal was to design a high-speed transport protocol (called XTP for Xpress Transfer Protocol [CHES88a, CHES89, PEI88a,b, PEI89]) and then implement that protocol in VLSI hardware. XTP is the high-performance protocol designed for the Protocol Engine Project which combines the classic transport and network layers into a single layer denoted as the "transfer" layer. XTP and the protocol engine have been in development for over two years, and continue to evolve as the XTP definition stabilizes toward a VLSI implementation. The current definition is XTP Revision 3.4, dated July, 1989.

This section presents a brief overview of XTP, followed by some of the specifics within the XTP definition. This is not a tutorial, however, and we do expect some fluency with the current definition of XTP. The primary purpose of this overview is to explain what XTP does to handle flow and rate control, and the history behind the *sort* field so that the presentation of the stabilized scheme for the *sort* field in XTP Revision 3.4 will have meaning.

6.1. Protocol Overview

There are three main goals driving the design of XTP. The first is that XTP must maintain full functionality. The protocol must provide a transparently reliable service; XTP is designed to provide the classic functionality of the transport and network layers. XTP is the most useful set of functions consistent with a simplified implementation. Second, XTP will provide real-time performance. The most efficient use of an underlying medium is to stream data to it at media rates. XTP is designed so that it will complete all protocol processing for an inbound or outbound packet in an amount of time equal to the transmission of the packet on the media. Finally, the protocol engine which will be the hardware implementation of XTP must provide integrability. The protocol engine itself should achieve the desired functions and performance in a small number of VLSI packages.

By design, XTP's basic service is connection-oriented (i.e., virtual circuits). Datagrams are treated as short-lived connections. The fundamental philosophy of the underlying connection-oriented service is based on the observation that a receiver is inherently more complicated than a transmitter. Because the receiver must perform checks and tests, system performance is heavily dependent upon the receiver's performance. XTP attempts to simplify the receiver as much as possible so that it is a very high performance engine. An example is that receivers do not generate response messages unless specifically commanded to do so; this reduces backtalk and generally improves performance. XTP's design strategy is to start with a clever receiver architecture. The design of the transmitter follows as a complement to that of the receiver. The transmitter's tasks are to build output packets and control the receiver with command codes within those packets.

Because the protocol engine is designed to provide the transport (transfer) layer user with data rates comparable to those found at the media access layer, data buffering is intentionally quite limited — the protocol engine must accept, evaluate, and buffer one packet in the time it takes for a second packet to arrive. To achieve the high data transfer rates needed, accepting a packet consists of a series of steps. This sequence of events is described below and shown in Figure 10.

(1) Translation. As the packet header arrives, a *key* in the header is used to look up state vector information in memory.

(2) Load. The result of the address translation is used to load the appropriate state vector into the protocol engine. If no connection is active (as would be the case with a datagram or with the first data packet of a connection), a new state vector is constructed.

(3) Evaluate. Sequence numbers are checked and the packet is determined to be in order (so processing continues) or out of order (so the packet is dropped).

(4) Buffer. Incoming data is buffered pending the next accept/reject decision.

(5) Commit. After the protocol engine computes the frame check sequence, the receiver either discards data if the CRC is incorrect or else accepts the data and updates the connection’s state vector.

From the outset, the design of the protocol engine hardware is intended to scale from medium speed (10 Mbps for Ethernet) to high speed (100 Mbps for FDDI) to very high speed (1 Gbps for networks now operating in the laboratory). The initial target is to interface the protocol engine to the Ethernet chip set, and afterwards to the FDDI chip set.

A block diagram of the protocol engine is shown in Figure 11. The conceptual design is that the protocol engine, running XTP, would be reduced to three, four, or five VLSI chips, and that this collection would provide the transfer layer services. The protocol engine would in turn interface to FDDI or Ethernet chips for its datalink and physical layer services.

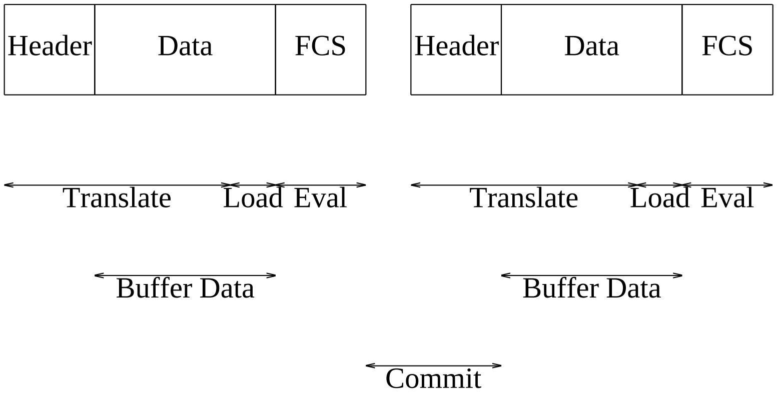

Figure 10 — Time Diagram of XTP Sequence of Events

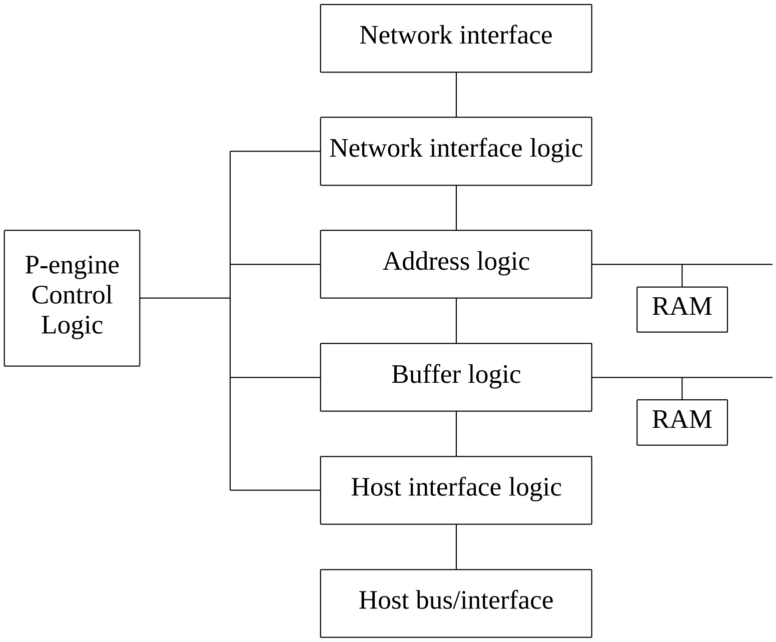

Figure 11 — Block Diagram of the Protocol Engine

A pair of protocol engines can be used back-to-back as a bridge across same-type address domains or as a router across dissimilar address domains. This latter option has significant consequences with regard to overall protocol design, buffer management, and flow control. Figure 12 shows how protocol engines can be used to interconnect networks. Information arriving from, say, an FDDI network is accepted and buffered by one protocol engine, then delivered to the system bus which interconnects protocol engines. From there a second protocol engine accepts the message and transmits it on another, possibly dissimilar, network. Each of these networks has its own implementation of the transport layer and below, both possibly XTP implementations. Packets entering from one network destined for the other must actually leave the protocol stack attached to the source network before entering the protocol stack attached to the destination network. This performs a protocol conversion; this would be necessary between a hardware implementation of XTP and a software implementation.

Since XTP is a connection-oriented protocol, there are state machines and variables associated with each connection. The set of these state variables at one end of a connection are referred to collectively as a *context*. A connection is established between two user processes, each of which acts or reacts to the changes within the context on that end. XTP exchanges packets between the ends of the context, one format for data exchanges and one format for control information exchanges. The data packet contains header and trailer information, such as an address key, a sequence number, a packet command code and a type code. The command code tells the receiver to buffer the user data, generate a response message, or establish/remove

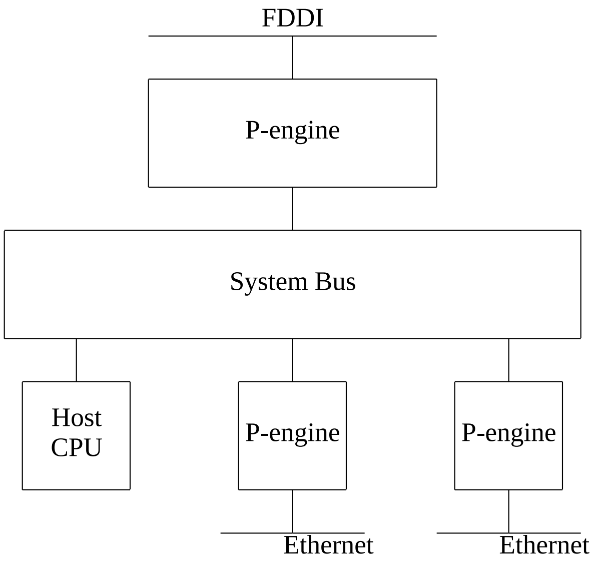

Figure 12 — Network Interconnection Using Protocol Engines

a connection context. The sequence numbers are the actual data byte count on that context. Control packets contain flow, error, window, message, and timestamp information important to the protocol but not needed with every packet.

6.2. Protocol Specifics

Some of the state variables within a context control the flow of data between XTP entities, and some control the rate at which that data should flow. Flow control is a method of restraining the volume of information that may be sent to help avoid congestion at the transport connection end points and on the network connection. Rate control is the technique of limiting the rate at which a sender is allowed to transmit data, usually by enforcing a time delay after each packet and/or a time delay after each burst of packets. Rate control can limit congestion in the center of a network, i.e., at gateways, as well as limiting overruns at receivers.

A typical use of flow control is when traffic is heavy and continuous, or when there is intensive multiplexing. Proper use of flow control can optimize response times and resource utilization. One method of imposing flow control is to use a *sliding window* protocol, where the window size indicates the number of PDU’s that the receiver is allowing the sender to transmit. By varying the size of the window, the sender’s transmissions can be throttled according to the resources available in the receiver.

In XTP, a receiver typically gives a do-not-exceed packet sequence number to a remote host. This parameter is called *alloc*, and it constrains the sender from introducing more data than the receiver’s buffers can accept. The *alloc* field appears in the control packet, and its value is communicated from the receiver to the transmitter. The value of *alloc* is one greater than the highest sequence number that the receiver will accept.

The *dseq* field is in the common trailer. It is the sequence number of the next byte to be delivered to the destination application process; it is one greater than the sequence number of the last byte delivered to the destination. All bytes with sequence number less than *dseq* have been successfully transferred to the destination client. The *rseq* field is in the control packet. It is the sequence number of the first byte not yet received from the network. All bytes associated with sequence numbers less than *rseq* have been input and buffered by the receiving XTP process, but may not have been delivered to the destination process yet. Thus, *rseq* is one greater than the largest consecutively received data byte sequence number. The relationship between *alloc*, *dseq*, and *rseq* is:

$$dseq \leq rseq \leq alloc.$$

XTP also provides a reservation mode for bulk transfer. The receiver is requested to provide an allocation window that represents only the amount of space available in locked down application receiver buffers. This is necessary on very high speed networks to prevent queue overflows. This guarantees that queues can be drained into receive buffers.

XTP is designed to be used on several different scales. There will be many who employ a software version of XTP and some who use the hardware protocol engine. Underlying physical layer signaling speeds may vary from installation to installation. Because they have XTP in common, they can all be interoperable. However, interfacing an FDDI network to, say, an Ethernet could cause severe problems in a traditional transport protocol. If multiple packets were directed from the FDDI network to the Ethernet, the speed mismatch would undoubtedly cause buffer congestion in the intervening gateway, leading in turn to buffer exhaustion, a closed sliding window, and eventual retransmission of lost packets. While no protocol can make the 10 Mbps Ethernet accept data at the 100 Mbps rate of FDDI, XTP does regulate the rate of data by allowing the protocol engines, acting as routers or gateways, to participate in the rate control decisions. Unlike ISO Transport Class 4 or TCP, in which rate control is approximated by flow control which operates end-to-end, XTP rate control operates between protocol engines. This allows XTP to do a much better job in regulating rate between networks of differing capacity.

XTP gateways assign a maximum packet rate to each new route, applied to all traffic from the source along the route. This is a maximum do-not-exceed bandwidth, or packet rate, to each unique source/destination path throughout the internet. Connections sharing a particular route must divide the assigned rate to avoid a surge at a gateway. Gateways along a route may regulate rate according to observed load and available resources. This mechanism for rate control is seen as more efficient than rate control via sliding window flow control scheme, resulting in less work for gateways. An XTP router can administer rate control based on the number of source/destination pairs, not the number of active connections, which would be much higher.

The rate control parameters are *separation* and *credit*. The value of *separation* is used to guarantee that the slow receiver has sufficient time between back-to-back packets to complete protocol processing on each data packet before the arrival of the next data packet. Changing the value of *credit* allows the router to dynamically control the flow into the router so as to avoid overwhelming it with requests. The sender is allowed to send an amount of data not to exceed the amount limited by *credit* in a 1/60 second time period with minimum spacing between packet of *separation* microseconds. This specifies interpacket spacing as well as aggregate rate, which is useful when a hardware implementation is communicating with a software implementation since it is unlikely that the software implementation could handle back-to-back packets. Also, in hardware to hardware communications, back-to-back packets may be lost if it takes a too long to deliver a burst to the host.

Both the *credit* and *separation* fields are sent in a control packet. Their values are communicated to the sender, and, together with the flow control parameters and local allocation policies, constrain the sender in order to avoid as many problems during the time that the context is active as possible.

Some local allocation parameters include *robin* and *oqmax*. The *robin* parameter to a context imposes a round-robin style of satisfying active context's request for service. The *oqmax* parameter limits the maximum size of the output queue per context. The maximum output burst for any particular context is the minimum of *credit* and *robin*.

6.2.1. Sort Field

Like many transport protocols, XTP redefines "real-time communications" as being "very fast communications." More than any other protocol, XTP is likely to achieve the goal of being very fast. Still, that is not the same as ensuring that no application process will miss its real-time deadlines. A fast, light-weight protocol increases the probability of acceptable real-time performance, but does not guarantee it. The later revisions of the XTP definition have attempted to address the issue of making XTP responsive to real-time needs. The so-called *sort* field is a result of these attempts. Over the last several revisions of the XTP definition, the *sort* field has had an interesting evolution, which is presented here as historical perspective before endeavoring to explain its current syntax and semantics.

Prior to XTP Revision 3.25, priority was a local issue; the interface between the user and XTP (e.g., the *control block*) defined a priority field which communicated the priority of the operation requested by the control block. The real-time community, including the Navy, with their interest in XTP for SAFENET II, and industrial and academic constituents, expressed concern for the lack of an end-to-end approach to scheduling message processing.

6.2.2. XTP Revision 3.25

Revision 3.25 included an *order* field within the address segment of the initial information packet, and a *sort* field within the control block. The *order* field was 32-bits long and provided a way to specify end-to-end priority or other system-dependent concepts. It was sent through the address segment to the receiver where XTP would deliver the *order* value to the destination host. It was to be interpreted at the destination by convention established between end systems. XTP was not to know the semantics, nor attempt to interpret the semantics of the *order* field. Each XTP implementation was expected to have at least one output queue, where output refers to the host to XTP exchange. Each output queue would be associated with a 32-bit *sort* value. The output control block would be delivered to XTP, where the *sort* value in the control block would select an output queue. These queues would operate on a priority basis; a control block with lower sort value would be serviced before one of higher sort value when the next control block was being selected. The XTP user could alter the *sort* and/or *order* values in the control block by using the MODIFY operation on the busy control block.

This mechanism was promising insofar as the priority could be communicated via the *order* field between end systems. However, the *order* value was to be sent once, within the address segment of the initial information packet, and not sent again over that context unless the context were to send another address segment, which would likely be unnecessary. The *order* value, however, would be interpreted by the end systems, remained constant throughout the entire life of the context. It was the *sort* value which could change. The idea of prioritized output queues solved the problem of scheduling messages where the most is known about their importance — at the entry point. However, since this value was not communicated, the destination system had no idea how important the source system deemed that packet, and thus the special treatment at the source would be lost to the destination.

6.2.3. XTP Revision 3.3

Revision 3.3 dropped the *order* field and used the *sort* field as the end-to-end method of communicating priority. The job of the old *sort* field, that is, to select an output queue, was taken over by a new field called *qnumber*. This new field would do exactly what the old *sort* field did, only the name is more indicative of its purpose.

XTP retained the term *sort* instead of priority since priority has so many connotations acquired from classical system methodology, and *sort* suggests that the packets are being arranged according to some schema. The *sort* value would denote a computed value used to determine a schedule for the processing of the packets. Next generation system are expected to label tasks and messages with values which appear as priority but which are actually sophisticated functions capturing concepts of urgency and importance and high-resolution time. This is not adequately solved by only a few bits of priority at the transfer layer. XTP would provide 64-bits in the *sort* field for current and future use.

The *sort* field would be located within the information segment of an information packet. It would be neither header data nor user data; it would be, rather, out-of-band information exchanged between end systems as a means for scheduling the packets at both ends of the route. Like with the *order* value, *sort* would not be interpreted by XTP itself. In some sense, the *sort* value would be the output queue label and the *qnumber* would be the implementation-dependent queue identifier used for selecting the output queue for the control block on systems that provide priority queues on a host to XTP exchange.

It would be the responsibility of the user of XTP (e.g., a real-time operating system) to map from *sort* values to queue numbers. Different applications would define different sort-to-queue functions. These functions must be duplicated in the gateways so that incoming sort values can map packets to outgoing queues.

The mechanism defined for XTP Revision 3.3 differs from the one defined for Revision 3.25 in few ways. The names of the field were changed around so that they better indicated their use. The *sort* field was increased to 64 bits and moved to the information segment. It could be sent at any time as long as the appropriate flag bits were set to indicate that the first 8 bytes of the information segment were being used for the *sort* value.

This mechanism for scheduling the processing of packets within XTP is championed mostly by visionaries working in future real-time operating systems research [JENS89]. The distributed computing system may incur communications resource congestion when the node processor speed and message generation rate become very high compared to the transport rate of the communications subsystem. During such congestion, transient queues build up at the entry and/or exit points of the communications subsystem. XTP, through mechanisms such as the *sort* field, would control the order in which messages waiting at a node for transmission are processed. This ordering is currently provided by small integers serving as priorities, but such information as time is not present. The *sort* field semantics would be defined by XTP's clients and not be interpreted by the XTP subsystem. The 64 bits would provide enough width to include absolute time measured in nanoseconds as well as other information such as importance. The value would be passed along with each message so that it may be used by the destination client for exit scheduling.

We feel that this mechanism for making XTP responsive to real-time needs is not quite sufficient. The *sort* field in Revision 3.3 was designed to provide some means of scheduling packets without constraining any particular user or implementation of XTP to that scheme. It was intentionally vacuous. XTP itself could only passively administer the discrimination policies of the user; the interpretation of the *sort* field was left to the agreement of the end systems. A system designer would create the policy, XTP would provide the mechanism. We feel that XTP should define both the mechanism and the scheme, but that the scheme should be flexible enough to apply to all types of networks, yet be potent enough to provide for each user's special needs, including the special needs of real-time.

6.2.4. XTP Revision 3.4

The Revision 3.4 *sort* field is a 32-bit field which associates a value to each packet. There are two mutually exclusive interpretations of the value within the *sort* field: it may be used as a 32-bit static priority or as a 32-bit clock time. The latter interpretation is designed for use in real-time system where deadlines are imposed. In both cases the increasing *sort* values correspond to lower priority. This is especially good for deadline driven communications where the difference of the deadline and the current time approaches zero, indicating that the message is approaching the highest priority.

The *sort* field is enabled by a SORT bit in the *options* field of the XTP header. There is also a DEADLINE bit which, if set, indicates that the deadline interpretation is in use; otherwise, the static priority interpretation is in use. The deadline scheme dictates that the 32-bit *sort* field be used as a clock time with a resolution of 100 microseconds. This is largely to satisfy the current and projected timing requirements of real-time operating systems. An XTP implementation should not implement an expired deadline policy; this is considered the responsibility of an outside authority, such as a user process. XTP should alway deliver, or operate on, the next available *sort* value.

The homogenization of the two *sort* field interpretations is considered prohibitive. Thus it is assumed that a network based on XTP will either be using the static interpretation or the deadline interpretation, but never both at the same time. If the SORT bit is not set then the field is ignored and no policy is required. It is further assumed that the XTP-based hosts will have some access to a synchronized clock.

An XTP implementation will impose a preemptive priority queue discipline on both input and output. An application will supply an active context with the current *sort* value for output packets. This value will be inserted into each packet's header for use at the receiving side. All packets of the currently highest priority (lowest *sort* value) at the host are transmitted prior to the transmission of any packet of lower priority, subject to rate and flow control constraints. Only if a context is blocked by rate or flow control can a packet of lower priority be serviced before those of higher priority. As soon as the flow or rate control constraint is off, however, the XTP implementation must revert back to transmitting the highest priority packets. The receiving XTP implementation will copy the *sort* value from the *sort* field in the packet header and place it into the local context. This value is then used to schedule the order in which data is delivered to the applications.

The advantage to having a context hold the *sort* value for its messages is that the next message to service is easily found by ordering the contexts. Also, the XTP user has control over the *sort* value by updating the value dynamically, while the message awaits service. Unfortunately, this only works well with the static priority approach. All messages in a context cannot be expected to have the same deadline, and hence the same *sort* value. This problem will cause the text of Revision 3.4 to be changed.

It may be naive to assume that a network, real-time or not, will use only the static priority or the deadline scheme, but not both. Gateways and routers will undoubtably have some traffic with deadline bit set, some without, so it will have to homogenize the two schemes. This is a very difficult task, as described in Section 5.2. General homogenization may not be possible. Even with the two simple interpretations, the question of homogenization is so open that XTP has declared that the two interpretations will be used in mutually exclusive installations.

The nearest deadline first is the easiest scheduling algorithm for deadlines. It is also troublesome. There are instances when a message must get to its destination, even if it misses the deadline. There are other instances when it is not only inefficient but harmful if a message with a missed deadline is delivered. This may be why XTP has declared that it makes no policy for missed deadlines, and always tries to deliver messages unless instructed not to by a higher authority. Also, nearest deadline first probably does not accurately describe all deadline traffic; there is at least a residual importance associated with each message, regardless of how near its deadline is.

Section 5.4 describes a scheme based on priority classes, where each class has a semantic interpretation by the transport layer implying that the transport layer becomes an active participant in the discrimination of the packets it handles. The preemptive priority queue discipline defined for XTP Revision 3.4 does not nail down the power of the preemptive class; that is, that, with respect to transport level resources, it fatally remove all obstacles in the effort to ensure the lowest possible latency. The expedited class would bypass the flow control, so never would it be subject to priority inversion except when waiting for a cleanpoint. There are no specific sort values with these powers. Furthermore, it is not clear exactly what the definition of a preemptive queueing discipline is from the XTP Revision 3.4 document, and when packets may be preempted. Unless there is some notion of fatal preemption of packets, there will be priority inversion at the transport layer. We feel that the scheme described in Section 5.4 is implementable within XTP without any major restructuring of the protocol algorithms. Preemption can be handled by XTP on only those resources that XTP has control over, like certain buffer spaces and the use of the packet deliver protocol (most likely FDDI MAC). This scheme is not dependent on preemption at the very low layers of the protocol suite; only within XTP itself. We feel that including timestamps becomes less necessary when the protocol itself can actively participate in latency control. Furthermore, many of the problems identified with the current *sort* algorithms can be eliminated with this scheme.

In general, using the dynamic priority features of XTP will be difficult. We suggest a great deal more explanation, perhaps as an appendix. We believe that implementers would run the gamut from being overly simplistic to overly complex in trying to implement a *sort* field with the guidance of Revision 3.4 only.

7. Conclusions

This is an ongoing research project for which the answers to many of the open questions still need thought. We are attempting to make a new communications protocol responsive to the needs of its users, especially the very constrained needs of real-time systems. Without designing the communications subsystem from the bottom up, endowing each component with the attributes that are required by the importance abstraction, this is a truly difficult task. Yet we still wish to give current transport protocols the mechanism to do the best they can to meet the needs of their users. We are identifying the aspects of these transport protocols and the communications subsystem in which they are a part that are hindering this goal, and those characteristics that are useful in simplifying the task. We have presented here our initial abstraction of importance, how importance can be related to communications subsystems, and some ideas and schemes which probably will not prove to be the ultimate encoding schemes but are useful and instructive toward gaining closure on our goal.

As a case study of a transport layer protocol which is being developed to meet a wide range of commercial and military needs, we presented a brief overview of XTP and some of the specifics of the protocol. XTP is unique in that there is a concerted effort toward making XTP responsive to real-time needs. The definition of XTP changes and evolves; during these changes several ideas about how to make XTP responsive to special needs as well as usable for common communications have been explored. We presented a history of these ideas and discussed their advantages and disadvantages. Based on the importance abstraction and realistic expectations of communications subsystems, we have proposed a scheme for use within XTP that provides latency control for very special packets, even at the expense of throughput. Each of the scheme's three classes of service, normal, expedited and preemptive, has a relative ordering in its priority space as well as semantics within XTP itself. This is to say that XTP can actively aid in the reduction of latency for the most important packets, which is tantamount to the claim that the most important packet will have the minimum delay.

ACKNOWLEDGEMENT

This work was sponsored by the Naval Surface Warfare Center, Dahlgren, Virginia, under contract number N60921-87-D-A315 task B062, and the Southeastern Center for Electrical Engineering Education. Our technical director at NSWC is David Marlow.

REFERENCES

CHES87a Chesson, G., "Protocol Engine Design", USENIX Conference Proceedings, Phoenix, Arizona, June 1987.

CHES87b Chesson, G., "The Protocol Engine Project", Unix Review, September 1987.

CHES88a Chesson, G., "XTP/PE Overview", *13th Annual Conference on Local Computer Networks*, Minneapolis, Minnesota, October 1988.

CHES88b Chesson, G., "XTP Design Notebook: Message Priority", *Transfer*, Vol. 1, No. 6, November and December 1988.

CHES89 Chesson, G., "XTP/PE Design Considerations", IFIP WG6.1/6.4 Workshop on Protocols for High-Speed Networks, Zurich, Switzerland, May 1989.

GORA86 Gora, W., Herzog, U., and Tripathi, S.K., "Clock Synchronization on the Factory Floor", *Proceedings of the Workshop on Factory Communications*, National Bureau of Standards, pp. 87-108, March 17-18, 1987.

HWAN87 Hwang, K.W., Tsai, W.T., and Thuraisingham, M.B., "Implementing a Real-Time Distributed System on a Local Area Network", (Abstract), *Proceedings of 12th Conference on Local Computer Networks*, pp. 142, October 5-7, 1987.

JENS85 Jensen, E.D., Locke, C.D., Tokuda, H., "A Time-Driven Scheduling Model for Real-Time Operating Systems", *Proceedings of the Real-Time Systems Symposium*, December 3-6, 1985, pp. 112-122.

JENS89 Jensen, E.D., "SORT", Comments to the Technical Advisory Board, PEI 89-91.

LELA85 Le Lann, G., "Distributed Real-Time Processing", *Computer Systems for Process Control*, pp. 69-90, 1985.

MIRA83 Mirabella, O., "Real Time Communications in Local Area Network Environment", *Proceedings of MELECON '83*, A1.07 Vol. 1, 1983.

PEDE88 Peden, J.H., and Weaver, A.C., "The Utilization of Priorities on Token Ring Networks", *13th Annual Conference on Local Computer Networks*, Minneapolis, Minnesota, October, 1988.

PEI88a "XTP Protocol Definition, Revision 3.25", Protocol Engines, Inc.

PEI88b "XTP Protocol Definition, Revision 3.3", Protocol Engines, Inc.

PEI89 "XTP Protocol Definition, Revision 3.4", Protocol Engines, Inc.

STOI88 Stoilov, E.I., "Coordination mechanisms in Local Area Networks for Real-Time Applications", *Computer Communications Systems*, IFIP, 1988.

STRA88a Strayer, W.T., and Weaver, A.C., "Performance Measurement of Data Transfer Services in MAP", *IEEE Network*, Vol. 2, No. 3, May 1988.

STRA88b Strayer, W.T., and Weaver, A.C., "Performance Measurements of Motorola's Implementation of MAP", *13th Annual Conference on Local Computer Networks*, Minneapolis, Minnesota, October 1988.

STRA88c Strayer, W.T., Mitchell, M., and Weaver, A.C., "ISO Protocol Performance Measurements", *ISMM International Symposium Mini and Microcomputers*, Miami Beach, Florida, December 1988.

STRA88d Strayer, W.T., Weaver, A.C., "Evaluation of Transport Protocols for Real-Time Communication", *University of Virginia, Department of Computer Science Technical Report No. TR-88-21*.

ZNAT87 Znati, T., and Ni, L.M., "A Prioritized Multiaccess Protocol for Distributed Real-Time Applications", *Proceedings of the 7th International Conference on Distributed Computing Systems*, pp. 324-331, 1987.