Multimedia Synchronization Protocols for Broadband Integrated Services

Thomas D. C. Little, *Member, IEEE,* and Arif Ghafoor, *Senior Member, IEEE*

Abstract—**Synchronization of multiple data streams in time has been recognized as a significant requirement of future multimedia applications utilizing broadband communication technology. Although this requirement has been approached for single channel media such as packetized audio and video, a general solution has not been identified for the provision of synchronization of real-time stream traffic as well as preorchestrated stored data to support multimedia applications.**

In this paper, we propose protocols to provide synchronization of data elements with arbitrary temporal relationships of both stream and nonstream broadband traffic types. We indicate how the synchronization functionality can be carried out within a packet switched network and, accordingly, present a two-level communication architecture. The lower level, called the network synchronization protocol (NSP), provides functionality to establish and maintain individual connections with some specified synchronization characteristics. The upper level, the application synchronization protocol (ASP), supports an integrated synchronization service for multimedia applications. The ASP uses temporal relationships of an application's data objects and manages the synchronization of arriving data for playout. The proposed NSP and ASP are mapped to the session and application layers of the OSI reference model, respectively.

I. Introduction

RECENT developments in high-speed communications technology have resulted in interesting new applications using the integration of voice, video, and data. Fiber optics and broadband integrated services digital network (B-ISDN) provide bandwidth and delay characteristics necessary to support these new media. One such application is the distributed multimedia information system (DMIS). This system supports integration and coordination of audio, video, text, and numeric data originating from databases or live, real-world sources interconnected by high-speed networks [39].

One of the requirements of any system supporting time-dependent data is the need to provide synchronization of data elements which experience random delays during transmission and retrieval. In a DMIS, this problem is particularly acute since several streams originating from independent sources can require synchronization to each other, in spite of the asynchronous nature of the network. In addition to the synchronization of *live* data streams that have an implied temporal correspondence, synchronization is also required for *synthetic* relationships which impose arbitrary temporal relationships on stored data elements of any media, including audio, video, text, etc. [1]. For example, a single image can be arranged to playout in synchrony with a segment of speech at a multimedia workstation.

For supporting the transmission of time-dependent multimedia data, real-time communication protocols and operating systems are essential. These systems assign the utmost priority to the meeting of deadlines for a set of scheduled time-dependent tasks. For multimedia data, catastrophic results do not occur when data are not delivered on time. For these data, real-time deadlines consist of the playout times for individual data elements that can be scheduled based on a specified probability of being missed. The design of such a system to support time-dependent media must account for latencies in each system component used in the delivery of data, from its source to its destination. Therefore, specific scheduling is required for storage devices, processors, and communication resources.

Most work on multimedia synchronization has been published only very recently. This problem is considered an important requirement for multimedia applications [1]–[12], [14]–[24], [26], [27], [36], [39], [40]. The work in this area typically applies to live data communications [2]–[10], data storage systems [1], [11]–[17], or general distributed systems [1], [8], [18]–[22].

For data communications, synchronization has typically been applied to live data sources such as packet audio and video [2]–[10] and requires that channel capacity should not be exceeded over the duration of the session, except in bursts. These applications can rely on a point-to-point configuration since there is seldom any need to provide intermedia synchronization of multiple independent live sources. Synchronization at the destination provides smoothing of the packet stream and tracking of the source packet production rate. At the source, data sequencing can ensure equivalent delays for multiple streams for the individual point-to-point connections [23], [24]. Analytical models have been developed for the description of a packet voice receiver (PVR) which reduces the variance of the jitter introduced in a real-time voice

Manuscript received December 11, 1990; revised July 11, 1991. This work was supported in part by the New York State Center for Advanced Technology in Computer Applications and Software Engineering (CASE) at Syracuse University, Syracuse, NY.

T. D. C. Little is with the Department of Electrical, Computer and Systems Engineering, Boston University, Boston, MA 02215.

A. Ghafoor is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

IEEE Log Number 9102860.

stream over a packet network [2], [4]. Similarly, models have been developed which, given an error bound, establish the delay introduced to meet the error specification as applied to the analysis of packet video [5]. Feedback and adaptive schemes for tracking of transmitted video in an asynchronous network have also been proposed [6]–[8]. The goal of these earlier techniques is to reshape the distribution of arriving packets to reduce delay variance. This process is shown schematically in Fig. 1, where $p(t)$ is the delay density function and $w(t)$ is the reconstructed playout distribution.

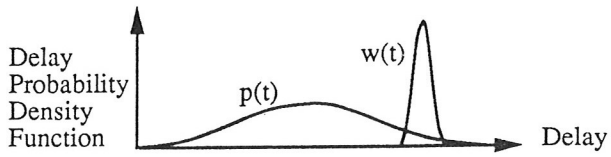

Fig. 1. Reduction of arriving packet delay variance.

With respect to communications protocols in the OSI framework, a mechanism is presented in [11] using the synchronization marker (SM) concept for indication of synchronization points and how this can be applied to the OSI reference model. For data sequencing, the *multimedia virtual circuit* (MVC) [23] has been proposed to guarantee the ordering and delay of synchronized data streams for single point-to-point connections, similar to the approach proposed in [24]. Nicolaou [18] identifies hierarchical levels of data elements for which different degrees of synchronization can be applied. In [1], a unified model is used to bridge the gap between synthetic and continuous synchronization. Karlsson and Vetterli [25] investigate the integration of packet video into the network architecture, noting that the OSI model was not designed to provide real-time transmission.

Synchronization is also an important consideration for stored-data applications [1], [11]–[17]. Audio and video require very large amounts of storage space and will exist, when not live, in a secondary storage. When data originate from storage, the system has more flexibility in scheduling the times at which data are communicated to the application. Since data are not generated in real-time, they can be retrieved in bulk, well ahead of their playout deadlines. This is contrasted with live sources, e.g., video cameras, that generate data at the same rate as required for consumption. Unlike live sources, stored data applications in a DMIS can require synchronization for data originating from independent sources, and simple data sequencing cannot provide intermedia synchronization.

In a distributed system, the distinction between the network and the storage devices is blurred. The requirements for the support of continuous media in such a system are the topic of [1], [8], [18]–[22], [26], [27]. Much of this work deals with the management of system resources to provide real-time data delivery. Typically, bandwidth and buffer space are managed via reservation to meet the statistical characteristics of time-dependent data streams.

A number of interesting problems remain to be solved to provide a general-purpose application synchronization service within the network architecture. For the network, problems include the specification of protocols to provide synchronization, the monitoring of data traffic for characterization of delays affecting synchronization, and the evaluation of such delays to provide synchronization. Problems for the application service include concurrent managing of multiple synchronized connections, locating multiple sources for composing distributed objects, and maintaining temporal information.

Our contribution to the synchronization problem is the development of a theoretical basis for arbitrary synchronization of multimedia data elements and the proposal of a set of protocols for providing this synchronization. We describe how synchronization can be performed within the network, and present a corresponding communication architecture. This work differs from earlier works in the following ways.

1) A single model is developed to handle both continuous and noncontinuous synchronization. Accordingly, a set of protocols is specified to enforce synchronization with a desired quality of service (QOS) with respect to delay, bandwidth, and probability of synchronization failure.

2) We assume that data streams can originate from multiple independent stored-data sources in addition to live sources, using multiple physical or virtual channels.

3) A concurrency mechanism for synchronization is proposed, such that streams can be synchronized to a single destination rather than to a single source clock. Most earlier work assumes that the source dictates the playout at the receiver (e.g., [6]). The problem with this notion comes with providing synchronization for two streams originating from independent sources.

4) We use *a priori* knowledge of data traffic characteristics to facilitate scheduling, when available, rather than providing on-line scheduling of dynamic input data based on its statistical nature (e.g., [19]–[21]).

5) We do not consider real-time *scheduling disciplines* for packets, prioritization, etc. (e.g., [20]); rather, we assume that such capabilities are available to the operating system or transport mechanism. The system is assumed to allocate various resources based on their availability, including channel capacity, buffer space, and priority [30].

6) The dynamic bandwidth requirements of a multimedia object are fit into finite resources (delay and channel capacity) rather than the resources dictating the feasibility of the application, i.e., the inherent burstiness of the object is smoothed via our algorithms (e.g., [26]).

7) Delays are traded off for the satisfaction of a playout schedule, even when the capacity of the channel is exceeded by the application.

The remainder of this paper is organized as follows. In Section II, we discuss theoretical aspects of intermedia synchronization for events and streams. In Section III, we describe arbitrary intermedia synchronization with Petri nets, their timing analysis, and propose algorithms to facilitate scheduling of synchronization for communication.

In Section IV, a set of protocols based on the proposed algorithms is described. Several examples illustrating their utility are presented in Section V. Section VI concludes the paper.

II. Intermedia Time Synchronization

The process of synchronization of data in the network consists of the evaluation of the temporal characteristics of the data to set up a connection, and the provision of real-time data transfer and anomaly correction when data are delayed or corrupted. In order to evaluate the temporal requirements of data elements, we must first understand the process of scheduling with respect to multimedia objects. In this section, we formalize the process of scheduling events and streams. Subsequently, in Section III, we describe the synchronization of hierarchies of temporally related data which we refer to as *nets*.

A. Data Synchronization

In order to properly synchronize some data element x with a playout time instant π, sufficient time must be allowed to overcome the latency λ caused by various processes such as data generation, packetization, transmission, etc.; otherwise, the deadline π will be missed. If we choose a time, called the *control time*, T, such that $T \geq \lambda$, then scheduling the retrieval at T time units prior to π guarantees successful synchronization. The *retrieval time*, or packet production time, ϕ, is defined as $\phi = \pi - T$. The case of synchronizing one event is simply the problem of meeting a single real-time deadline, which is a subset of the real-time scheduling problem. Fig. 2 shows the relationships between these aforementioned parameters. For streams of data, the multiple playout times and latencies are represented by $\Pi = \{\pi_i\}$ and $\Phi = \{\phi_i\}$.

The general scenario for synchronization of a data *stream* in a network is shown in Figs. 3 and 4. Here, elements of the data stream are generated at a source and experience some random delay with a distribution described by $p(t)$, before reaching their destination. They then require synchronization based on a playout schedule with deadlines π_i and playout hold times τ_i. Further, this problem can be extended from the cases of singular streams of packetized audio and video [2]–[10] to multiple streams and nonstream data (e.g., still images and text) as we show in Section III.

For a target packet loss characteristic, the control time T can be found for stream types using various approaches [2], [5], and represents the time required for buffering at the receiver to smooth variations in latencies λ_i. These variations in λ_i, as well as in τ_i, result in a short-term average or instantaneous need for buffering. For live sources, playout duration times are typically uniform ($\tau_i = \tau_j$, $\forall i, j$) and delay variation governs the buffering requirement. For the general case, we are presented with arbitrary streams characterized by π_i, λ_i, and τ_i, and would like to determine the amount of buffering necessary to maintain synchronization with a given probability of failure.

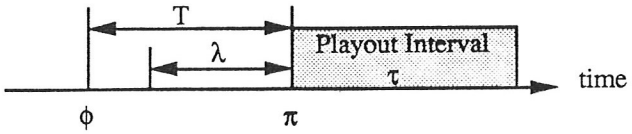

Fig. 2. Timing of single event synchronization.

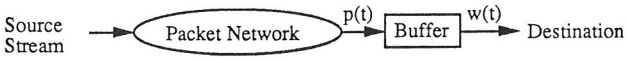

Fig. 3. Stream synchronization in a network.

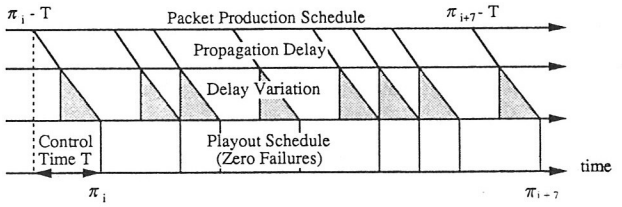

Fig. 4. Illustration of incurred packet delays.

In Fig. 5, we illustrate a cumulative probability distribution function for packet transit delay through a network, showing how the control time T can be evaluated from the end-to-end delay distribution at some probability of packet loss. The resultant T represents the end-to-end delay required to meet the specified loss characteristic. There is a tradeoff between this delay and packet loss since a large delay is not satisfactory for most real-time broadband streams (e.g., voice or video conversations), and packet loss introduces problems of data reconstruction (see Section III-E). Also, since the delay distribution can change with time, the loss specification can be violated given a constant T [7].

One of the assumptions in the earlier analyses is that the generation of packets is at a rate equal to the consumption rate as is typical for live sources. When we consider a database as the source of a packetized stream instead of a live source (e.g., camera), the packet generation can be greater than the consumption rate for some intervals and less for others, due to the adjustability of the data retrieval rate (see Fig. 6). In essence, data storage sources give us more freedom in controlling the packet production times. This is particularly important when we require concurrency in the playout of multiple media, as described in Section III.

B. Mulitple Stream Synchronization

Intermedia synchronization is the temporal coordination of multiple streams relative to one another, and requires concurrent streams to be played out at identical synchronization times π_i. Since it is desirable to transmit streams independently [28], [30], we decompose multiple streams into single ones for the purpose of identifying individual control times. However, data streams possessing the same schedule in terms of playout times may not nec-

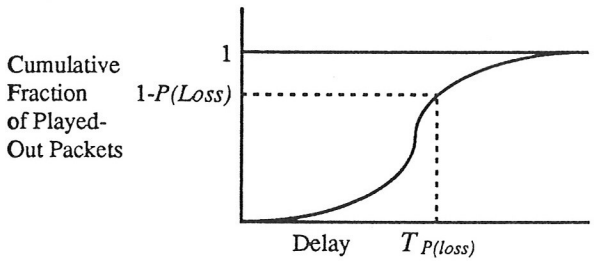

Fig. 5. Probability distribution for transit delay.

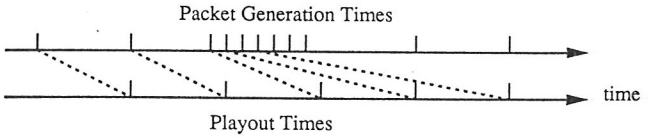

Fig. 6. Limited channel capacity: Adjustment of retrieval times.

essarily produce identical T's nor will they require the same buffering. This occurs since the streams can represent different classes of data in the network, and can have a different fixed delay component due to differing data size. To maintain intermedia synchronization, each stream must experience the same overall delay to avoid skewing. Adaptive schemes are unsatisfactory for this purpose since control times for multiple channels would necessarily adapt to their corresponding channel delays rather than to a single overall delay. Since intermedia synchronization is ultimately desired, to synchronize multiple streams, the control times of each stream must be accommodated, i.e., each stream must be delayed as long as the largest T. Nevertheless, the required buffering for each stream need not conform to the worst control time.

III. Multimedia Object (Net) Synchronizaton

In this section, we generalize the idea of synchronization to composite multimedia objects and present algorithms for generating schedules for object presentation and communication, which are the basis for the protocols of Section IV.

A. Temporal Specification Using Petri Nets

Suppose that a multimedia object is comprised of data elements x, y, and some stream S, and requires synchronization at tx_i and ty_i. These temporal relationships are difficult to capture with the stream semantics. A more general method to specify synchronization and the relationship between data elements uses a form of Petri net called the object composition Petri net (OCPN) [12]. The OCPN has been demonstrated to capture all possible temporal relationships between objects required for multimedia presentation [12]. The salient features of the OCPN are described as follows.

The OCPN is defined as a bipartite directed graph N_{OCPN}, specified by the tuple $\{T, P, A, D, R, M\}$ where:

$T = \{t_1, t_2, \cdots, t_n\}$ is a set of *transitions* (bars)
$P = \{p_1, p_2, \cdots, p_m\}$ is a set of *places* (circles)
A: $\{T \times P\} \cup \{P \times T\} \rightarrow I$, $I = \{1, 2, \cdots\}$ is a set of directed arcs (arcs)
D: $P -> Re$ is a mapping from the set of places to the real numbers (*durations*)
R: $P -> \{r_1, r_2, \cdots, r_k\}$ is mapping from the set of places to a set of *resources*
M: $P -> I$, $I = \{0, 1, 2, \cdots\}$ is a mapping from the set of places to the integers.

In this context, we assume that the resource component of the Petri net indicates a DMIS component which requires shared usage. For example, resource r_1 can indicate video data obtained from a specific database and communication channel. In this manner, network data traffic can be described based on its source and type, and can be associated with specific channel delay and capacity requirements. With the OCPN, we are free to choose the granularity of synchronization by assignment of time durations to the places. Associated with the definition of the Petri net is a set of firing rules governing the semantics of the model as follows.

1) A transition t_i fires immediately when each of its input places contain an *unlocked* token.

2) Upon firing, the transition t_i removes a token from each of its input places and adds a token to each of its output places.

3) After receiving a token, a place p_j remains in the *active* state for the interval specified by the duration τ_j. During this interval, the token is *locked*. When the place becomes inactive or upon expiration of the duration τ_j, the token becomes *unlocked*.

In Fig. 7(a)–(c), examples of three OCPN's are shown, representing sequential and concurrent presentations of objects (e.g., video frames and still color images, respectively), and an OCPN *template* for synchronous live audio and video. The OCPN template allows characterizing live sources with representative or worst-case playout durations and object sizes. These examples illustrate the strict and partial ordering that exists for various temporal relations among objects, and illustrates the possible flexibility in terms of deriving a retrieval sequence for the objects.

B. Playout Deadline Generation

Since an OCPN can easily represent synchronization of simple events and streams, we seek to determine playout times given the precedence relations and playout durations captured by the OCPN. These times then allow us to determine deadlines for data retrieval, and to identify the necessary buffering required. We propose the ***serialize-net*** algorithm for this purpose. An algorithm is also presented which decomposes a schedule into multiple playout subschedules formed by the elements unique to each traffic class or resource. This is called the ***resource-decomposition*** algorithm.

The ***serialize-net*** algorithm simulates an OCPN as follows. Beginning from the initial marking ($M(p_{\text{initial}}) = 1$ and $M(p_i) = 0$, $\forall i (1 < i \leq p_{\text{final}})$), the firing rules are obeyed, i.e., all enabled transitions fire, thus creating new

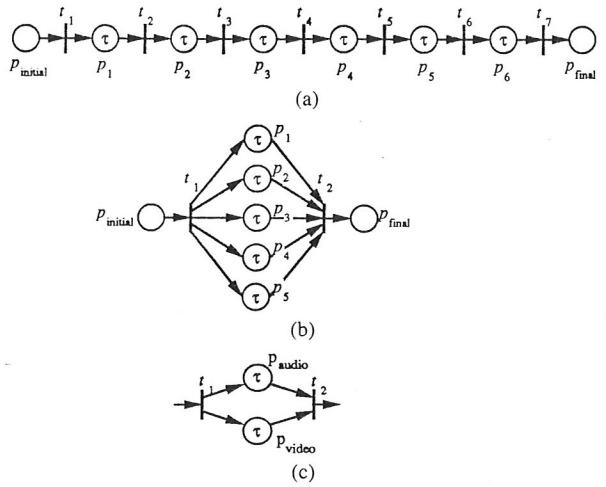

Fig. 7. (a) Sequential OCPN. (b) Concurrent OCPN. (c) OCPN template.

markings. As new markings are created, the start times of enabled transitions are identified as well as the start times of output places. The algorithm iterates until no new markings can be created, is only appropriate for marked graphs (a subset of Petri net models of which the OCPN is a member), and is guaranteed to terminate since the OCPN is acyclic [12].

Serialize-Net Algorithm:

$\pi_1 = 0$
new_marking = true
while *new_marking*
new_marking = false
  for each transition t_i in T
    if $M(p_j) > 1, \forall j : A(p_j, t_i) > 1$ **then**
      new_marking = true
      $M(p_j) = M(p_j) - 1, \forall j : A(p_j, t_i) > 1$
      $M(p_k) = M(p_k) + 1, \forall k : A(t_i, p_k) > 1$
      $st_i = \max(\{\pi_j + \tau_j\}), \forall j : A(p_j, t_i) > 1$
      $\pi_k = st_i, \forall k : A(t_i, p_k) > 1$
    end
  end
end

Table I shows the evolution of the markings for the OCPN of Fig. 7(a), assuming a video rate of 30 frames/second. For this example, the resultant schedule is $\Pi = \{0.0, 0.033, 0.067, 0.1, 0.133, 0.167\}$ seconds. For Fig. 7(b), $\Pi = \{0, 0, 0, 0, 0, 0\}$ seconds. These schedules represent the sets of deadlines for playout of data elements specified for each OCPN.

Once serialization is complete, a playout schedule can be decomposed into subschedules for each resource r_k, e.g., for each traffic class. The following ***resource-decomposition*** algorithm decomposes the different resource components, and creates multiple schedules representing subsets of the original:

Resource-Decomposition Algorithm:

for all places p_i in P
  if $R(p_i) = r_k$ **then** {identify places}
    $\Pi(r_k) = \Pi(r_k) + \pi_i$ {add to schedule}
    $i(r_k) = i(r_k) + 1$ {increment subschedule count}
  end
end

For example, this algorithm iterates through the set of audio/video deadlines in Π in the creation of subschedules Π_{audio} and Π_{image} representing the deadlines of distinct classes of data.

For nets, like streams, we need to determine the amount of buffering required for a specific probability of synchronization failure. Presentation concurrency can be serialized and resources can be decomposed, creating multiple schedules (one per resource). The amount of buffering and corresponding delay can then be determined for each subschedule. We have thus reduced the general synchronization problem to independent cases as described in Section II-B.

C. OCPN Scheduling and Control Time Generation

After the net has been serialized and decomposed into classes, a control time must be determined for each subschedule. In previous work, live sources have been studied in which the packet generation times simply track the

---

{start at time = 0}

{enabled transition}

{create new marking}

{transition time}
{playout time}

---

playout times with an introduced control time [2]–[5] or, if ϕ_i are the packet production times, then $\phi_i = \pi_i - T$, $\forall i$, where π_i are the playout times. The playout sequence is merely shifted in time by T from the generation time, as shown in Fig. 4. This constant timeskewing is possible since it is assumed that the channel capacity is never exceeded. However, for some stored-data applications, it is permissible for the OCPN playout schedule to specify retrieval of data exceeding the available channel capacity (e.g., see Fig. 8), and the previous scheduling policies are deficient [2]–[10]. Instead, we propose the deriving of a schedule that utilizes the full capacity of the channel by providing buffering at the destination based on *a priori* knowledge of the playout schedule Π.

1) Timing Analysis: We decompose the total end-to-end delay for a packet into three components: a constant delay D_p, a constant delay D_t proportional to the packet size, and a variable delay D_v which is a function of the end-to-end network traffic. D_p corresponds to propagation

TABLE I
GENERATION OF SCHEDULE II

new_marking	$p_j \mid M(p_j) = 1$	i	st_i (seconds)	π_k (seconds)
true	p_{initial}	—	—	$\pi_{\text{initial}} = 0$
true	p_1	1	0	$\pi_1 = 0$
true	p_2	2	0.033	$\pi_2 = 0.033$
true	p_3	3	0.067	$\pi_3 = 0.067$
true	p_4	4	0.10	$\pi_4 = 0.10$
true	p_5	5	0.133	$\pi_5 = 0.133$
true	p_6	6	0.167	$\pi_6 = 0.167$
true	p_{final}	7	0.20	$\pi_{\text{final}} = 0.20$
false	—	—	—	—

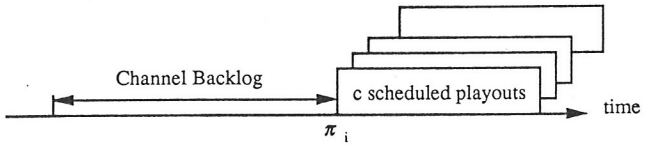

Fig. 8. Back-to-back arrivals for concurrency.

delays and other constant overheads (''pipeline'' delay). D_t is determined from the channel capacity C as $D_t = S_m/C$, where S_m is the packet size for medium m. D_v is a variable component determined by channel traffic. Therefore,

$$D_{\text{end-to-end}}(\text{packet}) = D_p + D_t + D_v, \quad \text{for individual packets for size } S_m. \tag{1}$$

A typical data object consists of many packets. Let $|x|$ be the size of object x in bits. The number of packets r required for object x is determined by $r = \lceil (|x|/S_m) \rceil$. The end-to-end delay for the complete object x is then

$$D_{\text{end-to-end}}(\text{object}) = D_p + rD_t + \sum_{j=1}^{r} D_{vj}. \tag{2}$$

Define control time T_i to be the skew between putting an object (packet) onto the channel and playing it out ($T_i = \pi_i - \phi_i$). Determination of a T_i is based on a selected probability of a late packet, called $P(fail)$. Given $P(fail)$ and the cumulative distribution function F of interarrival delays on the channel (of the random variable D_v), we can readily find T_i for a single packet using (1), as shown in (3). For an object comprised of multiple packets, T_i can be determined from (2), as shown in (4).

$$T_i = D_p + S_m/C + F^{-1}(1 - P(fail)), \quad \text{for individual packets, and} \tag{3}$$

$$T_i = D_p + r_i S_m/C + F_r^{-1}(1 - P(fail)), \quad \text{for single objects} \tag{4}$$

where $F^{-1}(p)$ determines the delay d for $p = F(d) = P(D_v \le d)$, and $F_r^{-1}(p)$ determines the delay d for $p = F_r(d) = P(\Sigma_{j=1}^{r} D_{vj} \le d)$. Since $F_r(d)$ requires a convolution on r random variables, it is computationally impractical for an arbitrary distribution on D_v when r is large. However, in this case we can use a Gaussian approximation for the distribution of a sum of r independent random variables, as justified by the Central Limit Theorem. Given the mean μ and variance σ^2 of the interarrival distribution, we can use the equivalent Gaussian approximation and determine the desired delay for the prespecified probability of synchronization failure.

For a set of *multiple* objects, it is possible to exceed the capacity of the channel and therefore we must consider their interaction. Depending on their size and playout times, transmission of objects is either back-to-back or introduces slack time. We define an optimal schedule for a set to be one which minimizes their control times. Two constraints determine the minimum control time for a set of objects:

$$\pi_i \ge \phi_i + T_i \tag{5}$$

$$\phi_{i-1} \le \phi_i - T_{i-1} + D_p \tag{6}$$

where i ranges over the set of objects, and corresponds to monotonically increasing deadlines. Constraint (5) describes the minimum end-to-end delay per object. It simply states that an object cannot be played out before arrival. This constraint must always be met. Constraint (6) describes the relation between a retrieval time and its successor when the channel is busy and accounts for the interarrival time due to transit time and variable delays (bandwidth constraint). This represents the minimum retrieval time between successive objects. The following theorem combines these results and describes the characteristics of an optimal schedule (see Fig. 9).

Theorem 1: An optimal schedule for any object i has the following characteristics:

$$\forall i,\ \phi_i \ge \pi_{i-1} - D_p \Rightarrow \phi_{i-1} = \pi_{i-1} - T_{i-1}$$

and

$$\phi_i < \pi_{i-1} - D_p \Rightarrow \phi_{i-1} = \phi_i - T_{i-1} + D_p .$$

Proof: Any object in the set can find the channel idle or with a backlog of items. If it is slack, the equality of (5) is optimal, by definition. Similarly, when the channel is busy, the equality of (6) is optimal. Clearly, a channel is slack when ϕ_i occurs after π_{i-1}, but it can also be slack when $\phi_i \ge \pi_{i-1} - D_p$ due to the pipeline delays. Therefore, the state of the channel, idle or slack, can be identified. □

We now show how to construct such a schedule as follows. Given D_v, D_p, D_t, C, S_m, π_i, $|x_i|$, we wish to construct an optimal schedule $\Phi = \{\phi_i\}$. We work backwards and begin by establishing an optimal retrieval time for the final mth object, i.e., $\phi_m = \pi_m - T_m$. The remainder of the schedule can be determined by iterative use of Theorem 1. The skew between playout and retrieval times for each object is now influenced by other objects using the channel, and is given by $debt_i = \pi_i - \phi_i$. The buffer space requirement at any given fetch time corresponds to $debt_i$, and is determined by the number and size of object retrievals completed between π_i and ϕ_i. Note that an opti-

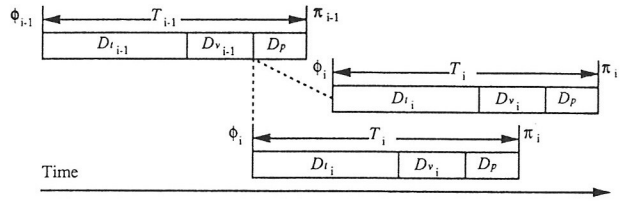

Fig. 9. Schedule constraints.

mal schedule need not be unique since multiple playout deadlines can share the same value (playout deadlines can form a partial order). The ensuing ***net-control-time*** algorithm computes the retrieval schedule as follows.

Net-Control-Time Algorithm:

$\phi_m = \pi_m - T_m$ {work backwards}
for $i = 0: m - 2$
  if $\phi_{m-i} < \pi_{m-i-1} - D_p$ {busy channel}
    $\phi_{m-i-1} = \phi_{m-i} - T_{i-1} + D_p$
  else
    $\phi_{m-i-1} = \pi_{m-i-1} - T_{m-i-1}$ {idle channel}.
  end
end

We can also determine the number of buffers required when using the playout schedule Φ, again noting that it is necessary to obtain an element prior to its use. For each fetch time ϕ_i, the number of buffers, K_i, required is equal to the size of the objects with elapsed playout deadlines between ϕ_i and π_{i-1}. The following ***buffer-count*** algorithm can be used for this purpose.

Buffer-Count Algorithm:

$K_1 = 0$
for $i = 0: m - 2$
  $K_{m-i} = 0$
  $j = i$
  while $(j < m - 1)$ and $(\phi_{m-i} < \pi_{m-i-1})$
    {count lagging playout periods}
    $K_{m-i} = K_{m-i} + |x_{m-j-1}|$
    $j = j + 1$
  end
end

From these algorithms we can determine the buffer use profile, ϕ_i versus K, and the maximum delay, *debt*, incurred by this buffering, which can be used for allocation of storage. Referring to the example OCPN's of Fig. 7, retrieval schedules and buffer use profiles are generated assuming the following conditions: $C = 45$ Mb/s, $S_m = 8192$ bits, $D_v = 50$ μs, and $D_p = 100$ μs. For Fig. 7(a), $\tau_i = 1/30$ s and $|x_i| = 2^{20}$ bits, corresponding to compressed video frames. For Fig. 7(b), $\tau_i = 20$ s and $|x_i| = 1024 \times 1024 \times 24$ bits, corresponding to color images. The results of the net-control-time and buffer count algorithms are:

Case (a): $\Phi = \{-0.0298, 0.0035, 0.0369, 0.0702, 0.1035, 0.1369\}$ s, $K = \{0\}$ Mb

Case (b): $\Phi = \{-3.56, -2.85, -2.14, -1.43, -0.71\}$ s, $K = \{24, 48, 72, 96\}$ Mb.

These results indicate that there is ample time to transmit the specified video traffic without buffering beyond the object in transit [case (a)], and that the image traffic requires 3.56 s of initial delay and up to 96 Mb of buffering at 0.71 s prior to initiating playout [case (b)].

From our discussion, we indicate that the solution to the synchronization problem lies in proper selection of control time for independent classes of data. However, we must consider practical considerations for determining network delay distributions and the handling of data lost in the network, which is described next.

D. Implementation of Schedules and Tradeoffs

In this section, we show how the schedules and control times can be used on a point-to-point basis. Basically, we can use the derived schedule Φ in two ways. First, the data source or transmitter can use it to determine the times at which to put objects onto the resource (channel). The receiver can then buffer the incoming objects until their deadlines occur. In this case, the source must know Φ, the destination must know Π, and the overall control time is T_1, the computed delay of the first element in the sequence. This scheme can be facilitated by appending π_i onto objects as they are transmitted in the form of time-stamps. The second way to use Φ is to determine the worst-case skew between any two objects as $T_w = \max(\{\pi_i - \phi_i\})$, and then to provide T_w amount of delay for every object. Transmission scheduling can then rely on a fixed skew T_w and the Π schedule; that is, transmit all items i such that $(t \le \pi_i \le t + T_w)$.

The first method minimizes both buffer utilization and delay since the schedule is derived based on these criteria. Furthermore, required buffer allocation need not be constant, but can follow the buffer use profile. The second method provides unnecessary buffering for objects, well ahead of their deadlines, and requires constant buffer allocation. However, it provides a simpler mechanism for implementation as follows. Consider a long sequence of objects of identical size and with regular (periodic) playout intervals. For this sequence, $T_1 = T_w$, and $\phi_i = \pi_i - T_w$, assuming the channel capacity C is not exceeded. In this case, the minimum buffering is also provided by the worst-case delay. Such a sequence describes constant bit rate (CBR) video or audio streams which are periodic, producing fixed-size frames at regular intervals. Rather than manage many computed deadlines/second (e.g., 30/second for video), a transmission mechanism more simply transmits objects based on sequence number and T_w.

When data originate from live sources, the destination has no control over packet generation times, and sufficient channel capacity must be present to preserve the real-time characteristic of the streams. In this case, the control time can be determined from the size, delay, and channel capacity requirements of a representative data object, e.g., a CBR video frame as described by an OCPN template, and only provides jitter reduction at the receiver. For variable bit rate (VBR) data sources, the data stream is statistical in size and frequency of generated objects, and we

supply a pessimistic worst-case characterization of the largest and most frequent VBR object to determine the control time. $\{\phi_i\}$ defines the packet production times, and playout times are determined by $\pi_i = \phi_i + T$, as done in previous work. This service is supported in our protocols by appending timestamps, in the form of individual deadlines, π_i, to the transmitted objects.

In the general case, a playout schedule is aperiodic, consisting of some periodic deadlines and others aperiodic. Typically, during class decomposition, these are isolated, and one of the two scheduling schemes above can be applied. If both exist in the same playout schedule Π (e.g., if classes are not decomposed), then the derived schedule Φ can be used as follows: choose a new control time T_E such that $(T_1 \leq T_E \leq T_w)$, and drop all deadlines ϕ_i from Φ such that $T_E \geq \pi_i - \phi_i$. The result is a culled schedule Φ reflecting the deadlines that will not be satisfied by simply buffering based on T_E. By choosing T_E to encompass a periodic data type (e.g., video), the burden of managing periodic deadlines is eliminated, yet aperiodic objects, requiring extensive channel utilization, can be dealt with using $\phi_i - T_E$, where $\phi_i - T_E > 0$.

When multiple delay channels are cascaded, e.g., channel delays plus storage device delays, we see that the end-to-end path capacity is reduced to the value of the slowest component. To facilitate scheduling of each resource, we can apply the derived schedule Φ of the first channel as the playout schedule of the succeeding channel, while moving towards the source. In this manner, each resource merely meets the schedule stipulated by the preceding resource in the chain; with the initial schedule being Π. For this scheme, however, to meet the same $P(fail)$, the failure probability used for computing Φ on each link must be divided between each component [27].

E. Synchronization Anomalies

Anomalies in synchronization for sequences of objects can be introduced due to inability to meet synchronization deadlines, by loss, and by corruption of data in the network. Suitable policies must be developed to remedy this situation and yet still attempt to satisfy ongoing deadlines of subsequent objects. Retransmission protocols are not applicable to real-time data since they introduce unreasonable delay. At the presentation device, a synchronization failure can result in a shortage of data to playout per unit time, resulting in a gap.

A shortage of data can be caused by differences in source and destination clocks. Typically, the destination clock tracks the source clock in a point-to-point configuration for audio and video sources [5]–[7]. However, a single destination cannot track multiple source clocks and also provide intermedia synchronization when the source clocks are independent. For the proposed protocols, we assume that the destination controls the clock rate of the sources either via a global clock, periodic drift correction [31]–[34], or by using feedback [6], [35].

Various approaches have been investigated for the reconstruction of a playback sequence with gaps, e.g., extending the playout time of the previous element [36], or interpolation of earlier values [37]. When gaps in a data sequence are ignored with respect to the playout rate, the loss of data elements when subsequent data are available advances the sequence in time, and can be corrected by slowing the playout rate until the schedule is correct [1], [7].

For the proposed protocols, the playout of arrived data elements is determined entirely by the derived schedule. Data are released for playout at the indicated deadlines. Although we do not propose a specific gap-compensating scheme, a suitable one can be chosen such as sample interpolation [37].

IV. Proposed Protocols for Synchronization

In this section, we propose a set of protocols for providing synchronization of objects as a network service which are practically implementable. We distinguish two levels of synchronization to support general-purpose network service and application service. At the network service level, a service interface is provided to support applications requiring synchronization of sequences of single-class data traffic. This interface takes sets of data elements and their playout schedules and provides a synchronized output. At the application service level, temporal information in the form of an OCPN is utilized to produce playout schedules for synchronized communication of multiple classes of data with a specified $P(fail)$. By decoupling the network and application components, the network service only requires knowledge of deadlines and traffic class rather than data source locations, temporal relations, and object composition sites within the network [39]. We propose two synchronization protocols corresponding to these levels: the application synchronization protocol (ASP) and network synchronization protocol (NSP). The relationship between these layers, the application, and the transport mechanism is shown in Fig. 10. Later, in Section IV-C, we indicate how these protocols are mapped onto the framework of the OSI reference model.

A. Network Synchronization Protocol (NSP)

To provide a general interface and functionality to support applications requiring time synchronization, we require that the protocol take as input a set of objects O_i, their playout times π_i, their aggregate start time, and the desired probability of synchronization failure. The protocol can then provide us with the following:

1) A predicted end-to-end control time T for the set of data elements.
2) A point-to-point data transport mechanism.
3) Resynchronization and anomaly correction.
4) Signaling from destination to source to control playout rate.

We now discuss individual components required to construct the NSP.

1) NSP Service Primitives: From an end-to-end perspective, the control time for a sequence Π depends on

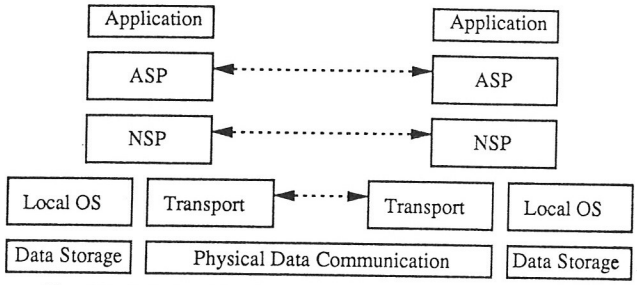

Fig. 10. Relationships between synchronization components.

the sequence itself, the interarrival distribution of consecutive packets, and any delays (e.g., a local control time) introduced in procuring the data at the source. If we can characterize these delays, then the control time can be determined using the procedure outlined in Section III-C.

For the NSP's, we assume that the network delays can be characterized by either measurement techniques [4], [29] or by guaranteed quality of service offered at access to the network [30]. For example, statistical data characterizing throughput, time delay, arrival rate, interarrival rate, and buffer occupancy can be collected using a hardware observational mechanism [28]–[30]. A network access mechanism can use these collected data for resource allocation decisions when setting up a connection. New connections can be accepted or refused based on available bandwidth or delay characteristics. When accepted, the access mechanism guarantees that the network is able to provide the requested level of service for a connection since system congestion is prevented by limiting additional connections. For synchronization purposes, a guaranteed level of service provides a well-defined characterization of the channel that the NSP can easily deal with.

We therefore assume, for the NSP, that the underlying network transport and multiplexing mechanism provides a characterization of interarrival distributions of consecutive packets through a service interface on a per-connection basis. At the time of connection establishment, given a class of traffic, source, and destination pair, channel capacity, packet size, and late packet probability corresponding to $P(fail)$, the access mechanism returns the channel delays and the mean and variance of the interarrival time of consecutive packets. We perceive this access mechanism to work equally well on network resources as well as operating system resources including database and storage subsystems. Accordingly, we define interface primitives for this service as follows.

$$[D_p, D_t, \mu, \sigma^2, \textit{resource-id}, \textit{r-avail}] = \textbf{\textit{reserve}}(\textit{class}, \textit{source}, \textit{dest}, \textit{P-late}, C, S_m)$$

$$\textbf{\textit{release}}(\textit{resource-id})$$

The ***reserve*** primitive requests network services of a specific class and probability of having late or lost arrivals. The returned parameters, D_p, D_t, μ, and σ^2 describe the channel delays, and the mean and variance of the interarrival time. If the resources are not available for this connection, the Boolean variable *r-avail* is returned false, indicating a refused connection. The *resource-id* simply identifies the reserved resources. Corresponding to the ***reserve*** interface primitive, a resource deallocation interface primitive is required. The functionality of this procedure is to release previously reserved resources for subsequent connections. With the parameters returned from ***reserve*** and a schedule Π, we employ the algorithms of Section II-C as internal NSP primitives as follows.

$$[T_E, \Phi] = \textbf{\textit{net-control-time}}(D_p, D_t, \mu, \sigma^2, \Pi)$$

$$[K] = \textbf{\textit{buffer-count}}(\Phi, \Pi).$$

The NSP supports a calling ASP entity by several service primitives. These primitives are defined for connection establishment, termination, and data transfer initiation (see Fig. 11). The primitives utilize the internal NSP primitives defined above and the ***reserve*** primitive of the transport mechanism to provide the network synchronization service. Note that the primitives can be invoked by the local process or by remote peers. For connection establishment, the ***NSP-estab-conn*** provides channel setup for synchronization service and is defined as follows.

$$[T_E, K, \textit{NSP-conn-id}, \textit{NSP-avail}] = \textbf{\textit{NSP-estab-conn}}(\textit{class}, \textit{source}, \textit{dest}, \textit{P-late}, \Pi).$$

The steps involved in connection establishment are as follows.

1) Receive (*class*, *source*, *destination*, *P-late*, Π).
2) If remote source (*source* $\neq$ *destination* = *my-site-id*) then:
   Invoke remote peer using
   $[T_E, K, \textit{NSP-conn-id}, \textit{NSP-avail}] =$ ***NSP-estab-conn***(*class*, *source*, *my-site-id*, *P-late*, Π).
   If resources are available (*NSP-avail* = *true*) then:
      Allocate and initialize K receive buffers.
3) If remote destination (*source*, = *my-site-id* $\neq$ *dest*), then:
   Reserve channel resources using
   $[D_p^c, D_t^c, \mu^c, \sigma^{2c}, \textit{resource-id}^c, \textit{r-avail}^c] =$ ***reserve***(*class*, *source*, *P-late*).
   If channel resources are not available ($\textit{r-avail}^c$ = *false*) then:
      Set *NSP-avail* = *false* to indicate lack of resources.
      Release the resources using ***release***($\textit{resource-id}^c$).
   Otherwise ($\textit{r-avail}^c$ = *true*):
      Derive channel schedule using
      $[T_E^c, \Phi^c] =$ ***net-control-time***$(D_p^c, D_t^c, \mu^c, \sigma^{2c}, \Pi)$.
      Determine channel buffer utilization using $[K^c]$ = ***buffer-count***(Φ^c, Π).
      Allocate and initialize K^c receive buffers for the channel.

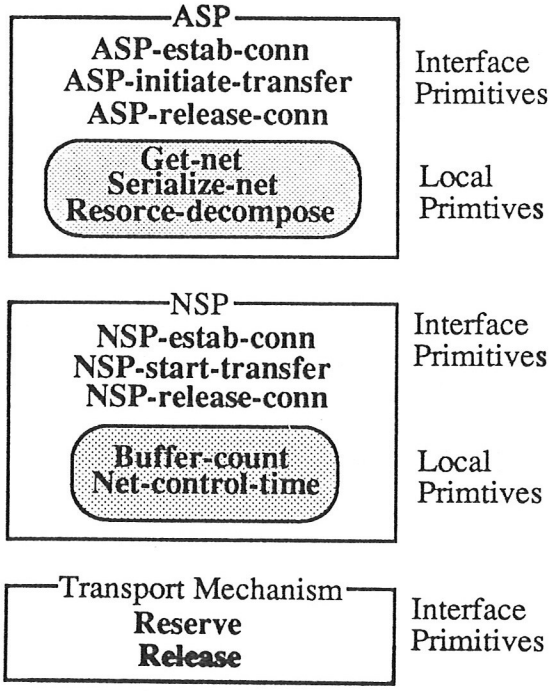

Fig. 11. Summary of ASP, NSP, and transport interface primitives.

Reserve storage resources using $[D_p^s, D_t^s, \mu^s, \sigma^{2s}, \textit{resource-id}^s, \textit{r-avail}^s] = \textbf{\textit{reserve}}(\textit{class}, \textit{source}, \textit{P-late})$.
If storage resources are not available ($\textit{r-avail}^s = \textit{false}$) then:
Set $\textit{NSP-avail} = \textit{false}$ to indicate lack of resources.
Release the resources using $\textbf{\textit{release}}(\textit{resource-id}^s)$ and $\textbf{\textit{release}}(\textit{resource-id}^c)$.
Otherwise ($\textit{r-avail}^s = \textit{true}$):
Derive combined storage schedule using $[T_E^s, \Phi^s] = \textbf{\textit{net-control-time}}(D_p^s, D_t^s, \mu^s, \sigma^{2s}, \Phi^c)$.
Determine storage buffer utilization using $[K^s] = \textbf{\textit{buffer-count}}(\Phi^s, \Phi^c)$.
Allocate and initialize K^s receive buffers.
Overall delay is the sum of storage and channel delays, $T_E = T_E^s + T_E^c$.
Set $\textit{NSP-avail} = \textit{true}$.

4. If local source and destination ($\textit{source} = \textit{dest} = \textit{my-site-id}$) then:
Reserve storage resources using $[D_p^s, D_t^s, \mu^s, \sigma^{2s}, \textit{resource-id}^s, \textit{r-avail}^s] = \textbf{\textit{reserve}}(\textit{class}, \textit{source}, \textit{P-late})$.
If storage resources are not available ($\textit{r-avail}^s = \textit{false}$) then:
Set $\textit{NSP-avail} = \textit{false}$ to indicate lack of resources.
Release the resources using $\textbf{\textit{release}}(\textit{resource-id}^s)$.
Otherwise ($\textit{r-avail}^s = \textit{true}$):
Derive storage schedule using $[T_E^s, \Phi^s] = \textbf{\textit{net-control-time}}(D_p^s, D_t^s, \mu^s, \sigma^{2s}, \Pi)$.
Determine storage buffer utilization using $[K^s] = \textbf{\textit{buffer-count}}(\Phi^s, \Pi)$.
Allocate and initialize K^s receive buffers.
Indicate overall delay $T_E = T_E^s$.
Set $\textit{NSP-avail} = \textit{true}$.
5. Return $[T_E, K, \textit{NSP-conn-id}, \textit{NSP-avail}]$ to the calling entity.

The propagation of computed control times and schedules is illustrated in Fig. 12. In this case, the computation of the retrieval schedule for the storage device (Φ^s) applies the retrieval schedule of the channel (Φ^d) rather than Π, as indicated in Section III-D.

The ***NSP-release-conn***(*NSP-conn-id*) provides an interface primitive for the ASP to terminate connections and release to resources. The primitive operates as follows.
1) ***release***(*NSP-conn-id.resource-id*).
2) Return.

Once a connection is established, the NSP must initiate data transfer based on the computed control times and perform the actual data transfer. An interface to the NSP for initiating data transfer and synchronization is defined as ***NSP-start-transfer***(*start-time*, *NSP-conn-id*), and operates as follows.
1) Receive (*start-time*).
2) Pass (*start-time-NSP-conn-id.*T_E) to peer ***NSP-start-transfer*** interface specified by *NSP-conn-id.source*.

Once initiated, the data transfer protocol sends data, via virtual circuits and the underlying transport mechanism, to the destination's buffers. The steps are:
1) At time = *start-time*, set transmit clock, *transmit_clock*, to zero.
2) $\forall i$, $\phi_i = \textit{transmit\_clock}$, send object i to the destination with appended playout deadline π_i.
3) $\forall i$, ($\textit{transmit\_clock} \leq \pi_i \leq \textit{transmit\_clock} + T_E$) send object i to the destination with appended playout deadline π_i.
4) Repeat steps 2 and 3 until Π is exhausted.

The protocol at the receiving site receives the data, providing buffering at a location specified by *NSP-conn-id.buffer*. Data are released to the application at times indicated by the appended deadlines π_i. Note that step 3) can be optimized for data with periodic playout times rather than the playout times of Π, as indicated in Section III-D. An example of the application of this protocol is described in Section IV-C. In the next section, we describe the services provided by the application level and show how they interact with the NSP.

B. Application Synchronization Protocol (ASP)

By partitioning the network and application levels, we provide two levels of utility for which different applications may be built. This division exposes possibilities for interesting issues such as multihop, multimedia composition, and load sharing in a homogeneous system [14]. Like the NSP, we propose a protocol for the application level (ASP). The NSP provides a general interface and functionality for many applications. On the other hand,

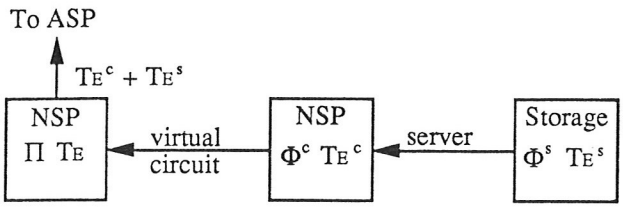

Fig. 12. Propagation of control times to the ASP.

the ASP provides specific services pertaining to the retrieval of complex multimedia objects from multiple sources across a network for playout at a single site. The ASP interface to the application takes as input a selected object representing the aggregation of complex multimedia presentation requiring synchronization, and return streams of distinct synchronized data traffic which can then be routed to specific output (workstation) devices for presentation. The interface also provides control over the quality of transmission service by specification of packet loss probability.

A typical application can use the synchronization service for stored preorchestrated presentations or for live sources such as videophones. For these applications, a temporal specification can be maintained in a database describing the object whether live or stored. Once an object is identified, the remaining steps required to retrieve and present the sought multimedia object are provided by the ASP. The overall processes needed for this purpose are as follows.

1) Retrieval of the temporal relationships describing the components of the complex multimedia object.

2) Evaluation of the precedence relationships of the OCPN, thereby creating a playout schedule.

3) Decomposition of the schedule into subschedules based upon the different traffic classes represented and the locations of stored data.

4) Determination of the overall control time required to maintain synchronization among the traffic classes through interaction with the NSP.

5) Initiation of data transfer.

Here we assume that all composition and synchronization is performed at the destination. It is also possible to perform composition at intermediate sites within the network, prior to transmission to the final destination [39].

For stored data, the process of identifying an object from a database is application-specific and consists of browsing or querying using database operations. Once an object is identified, the attributes describing its components and their temporal relationships can be retrieved in the form of an OCPN for schedule evaluation.

Given the net N_0 representing the time-ordering of object O, we can create a schedule Φ using the ***serialize-net*** algorithm of Section III-C. Recall that the ***resource-decomposition*** algorithm can convert this net into a set of subschedules based on the resource component, R, of the net. Each schedule $\Pi(p, q)$ represents the playout deadlines for all data elements of a resource r_j specifying the medium-location pair (p, q). Once the set of schedules is created from Π, the NSP can be initiated for the determination of control times for each r_j. In response to invocation, the multiple calls to ***NSP-estab-conn*** return multiple responses indicating the control times $T_E(p, q)$ and whether the required resources are available. The ASP must then determine the appropriate overall control time, and specify the time to initiate data transfer at all sites. If resources are available, the overall control time T_0 is found as max $(\{T_E(p, q), \forall r_j, (1 \leq j \leq n)\}$. T_0 represents the maximum control time of all paths from data sources to the single final destination.

Once T_0 is determined, the NSP sessions must be notified as to when to initiate their data transfer schedules. Transfer is initiated by passing the time to start to the ***NSP-start-transfer*** interface for each connection defined by r_j. The overall control time is added to the current local clock and passed to each interface as $clock + T_0$. Within the NSP, local control times are subtracted from this overall start time to determine local start times. Each NSP transfer protocol starts data transfer at a time equal to the overall start time minus its own control time. After initiation, multiple invocations of the NSP transfer data from sources to the destination with attached deadlines for playout. At the receiver (destination), the arrived data are released to the application process per the individual schedule, to be routed to the appropriate output device.

For applications requiring intermedia synchronization of live sources, e.g., audio and video of videotelephony applications, playout deadlines are generated on-the-fly from the stream of packets emanating from the source, and requires sufficient channel capacity to be available. The ASP and NSP can accommodate these sources as discussed in Section III-D: the worst-case object size and the shortest playout duration are applied to the ASP in the form of an OCPN template, which describes only the representative classes of data of the objects requiring intermedia synchronization. The ASP and NSP treat the OCPN template like any other object, returing a control time that provides compensation only for random channel delays for traffic specified by the template.

1) ASP Service Primitives: The ASP is developed in accordance with the requirements of both live and database sources. The following are the definitions of the various internal service primitives required by the ASP as specified by algorithms of Section III-B.

$$\{T, P, A, D, R, M\}] = \textbf{get-net}(object)$$

$$[\Pi] = \textbf{serialize-net}(T, P, D, A, M, L)$$

$$[\{\Pi(r_1), \Pi(r_2), \cdots\}] = \textbf{resource-decompose}(\Pi, R).$$

The ASP provides a service through application-ASP interface primitives. These primitives are ***ASP-estab-conn***, ***ASP-initiate-transfer***, and ***NSP-release-conn*** defined appropriately for connection establishment, data transfer initiation, and connection release. These primitives utilize the internal ASP primitives defined above and

the NSP to provide the application synchronization service. Each primitive is described below.

The ***ASP-estab-conn*** primitive sets up connections. It is defined as:

$$[ASP\text{-}conn\text{-}id,\ ASP\text{-}avail] = \textbf{\textit{ASP-estab-conn}}(object,\ P\text{-}late).$$

The steps involved in ASP connection establishment are summarized as follows.

1. Receive (O, *P-late*).
2. Identify the OCPN specification for the object using $[T_0, P_0, A_0, D_0, R_0, M_0, L_0] =$ ***get-net***$(object)$, where L_o is the set of storage locations of O's components.
3. Generate the playout schedule for the net using $[\Pi] =$ ***serialize-net***$(T_0, P_0, D_0, A_0, M_0)$.
4. Decompose the objects components into distinct traffic classes using $[\{\Pi(r_1), \Pi(r_2), \cdots\}] =$ ***resource-decompose***(Π, R_0).
5. For each resource r_k:
   Establish a session for the subschedule $\Pi(r_k)$ using
   $[T_E(r_k), K(r_k), NSP\text{-}conn\text{-}id(r_k), NSP\text{-}avail(r_k)] =$ ***NSP-estab-conn***$(class, my\text{-}site\text{-}id, L_0(r_k), P\text{-}late, \Pi(r_k))$
6. If resources are not availabe for any traffic class (any $NSP\text{-}avail(r_k) = false$) then:
   Set $ASP\text{-}avail = false$ to indicate lack of resources.
   For each resource r_k:
   Release the resources using ***NSP-release-conn***$(NSP\text{-}conn\text{-}id(r_k))$.
7. Otherwise (all $NSP\text{-}avail(r_k) = true$)
   Set $ASP\text{-}avail = true$ to indicate available resources.
   Determine the overall control time as $T_0 = \max(\{T_E(r_1), T_E(r_2), \cdots\})$.
8. Return [***ASP-conn-id***, *ASP-avail*] to the calling entity.

For ASP resource deallocation, ***ASP-release-conn*** (*ASP-conn-id*) primitive can be used by a calling application to terminate a session, such as the reception of the last object in a sequence. This primitive operates by invoking ***NSP-release***(*NSP-conn-id*) at both source and destination buffers for each *ASP-conn-id.NSP-conn-id*. For data transfer initiation, the ***ASP-start-transfer***(*ASP-conn-id*) primitive is defined and operates as follows.

1. Set *start-time* = *current-time* + *ASP-conn-id.*T_0.
2. For each connection, pass (*start-time*) via ***NSP-start-transfer*** interface, initiating data transfer.

The destination receives data in buffers set up by calls to the NSP and specified by *ASP-conn-id.NSP-conn-id.buffer*. The receiver function provides rerouting of the arriving data to the appropriate application presentation devices, beginning at *current-time* = *start-time* + T_0. At the source side, data is fed to the NSP buffers from the live input or database.

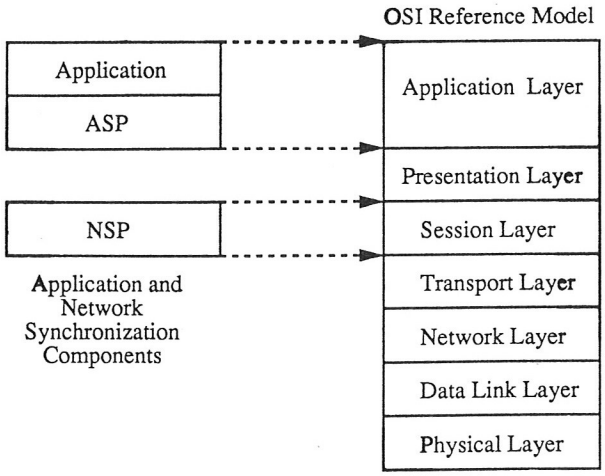

Fig. 13. Mapping of the ASP and NSP to the OSI reference model.

C. Mapping of the ASP and NSP onto the OSI Framework

The ASP and NSP provide two levels of synchronization. Fig. 13 shows a proposed mapping of these protocols to the OSI reference model, and their relationship to the network traffic monitoring and access mechanism. The NSP is mapped into the session layer (layer 5). This selection is made since the session layer deals with end-to-end connection establishment, authentication, binding, etc. The NSP synchronization service functions on a per-connection basis requiring the same level of service. This choice is consistent with other proposed and established placements of a synchronization service ([11] and [40], respectively), although other choices have been proposed as well. In [25], the presentation layer (layer 6) is suggested as the layer for resynchronization of packet video transmission services, while the session layer provides QOS tuning. In [18], both the presentation and session layers are represented as providing some component of synchronization service at a coarse granularity.

For the ASP, we note that cross connection required for intermedia synchronization is not mentioned in the OSI standards for layers 1–6 [11]. Therefore, the intermedia synchronization functionality of the ASP is mapped to the application layer. Other characteristics of the ASP, including maintenance of temporal information and database management of objects, is clearly a subset of the application level specification. The remaining lower layers of the OSI (layers 1–4) are responsible for data transport, routing, multiplexing, access control, traffic monitoring, etc.

V. Example: The Anatomy & Physiology Instructor

In this section, we consider an example to illustrate the use of the NSP and ASP. The *Anatomy & Physiology Instructor* is a multimedia application example requiring synchronization of stored data [12] in which a "student" may browse through information units (IU's) on various topics, reading text, viewing illustrations and audio/video

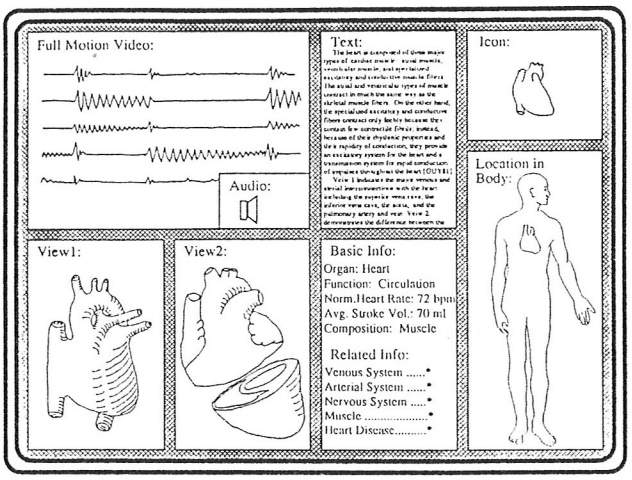

Fig. 14. The anatomy and physiology instructor.

presentations, as shown in Fig. 14. In addition, the user may perform queries or searches to locate specific topics by using a browsing map [12]. Synchronization is required for elements of text, image, audio, and video, within the context of an IU. Other functional requirements of the application include spatial manipulation of data (panning, zooming), temporal manipulation (stop, start), and database navigation. However, we assume that this functionality is provided by the application program and only describe the synchronization component.

There is no synchronization required between IU's as they are browsed in a random order. Within an IU, however, synchronization is required as described by the corresponding OCPN (see Fig. 15). In order to demonstrate the protocols, we assume a distribution of data storage over a network as shown in Fig. 16(a). In this case, individual data types are stored at distinct sites to provide data access performance optimization using specialized devices, e.g., real-time, multiple-head disks.

The overall process of retrieving the IU is described as follows. An object is selected by a user from the database and its temporal relationships are obtained in the form of an OCPN. The OCPN is used to create the schedule Π and the subschedules $\Pi(r_K)$; one for each medium-location pair. In this case, four medium-site pairs correspond to the audio, image, text, and video storage sites. The net of Fig. 15 stipulates a granularity for retrieval of each medium, e.g., the video is be synchronized in 5 s intervals, the audio at 10 s intervals, and the text and image at 10 s intervals. The derived subschedules have the form: Π_{video} = (0, 5, 10, 15, 20, 25) seconds, Π_{audio} = (0, 10, 20) seconds, Π_{text} = (0) seconds, and Π_{image} = (0, 10, 10) seconds. Based on the theory and algorithms proposed in Section III-C, retrieval schedules are derived for each subschedule using the ***NSP-estab-conn*** interface primitive, resulting in a set of control times: $T_{E_{\text{audio}}}$, $T_{E_{\text{image}}}$, $T_{E_{\text{text}}}$, and $T_{E_{\text{video}}}$. The maximum control time T_0 is found and returned to the ASP for generation of the overall start time. The resultant protocol interaction is shown in Fig. 16(b) where four NSP–NSP concurrent virtual connections are shown, each responsible for a specific data type and channel characteristic. At the ASP, the streams of data of each connection are routed to the appropriate presentation devices.

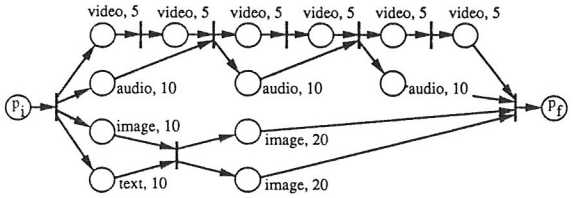

Fig. 15. OCPN for the anatomy and physiology instructor.

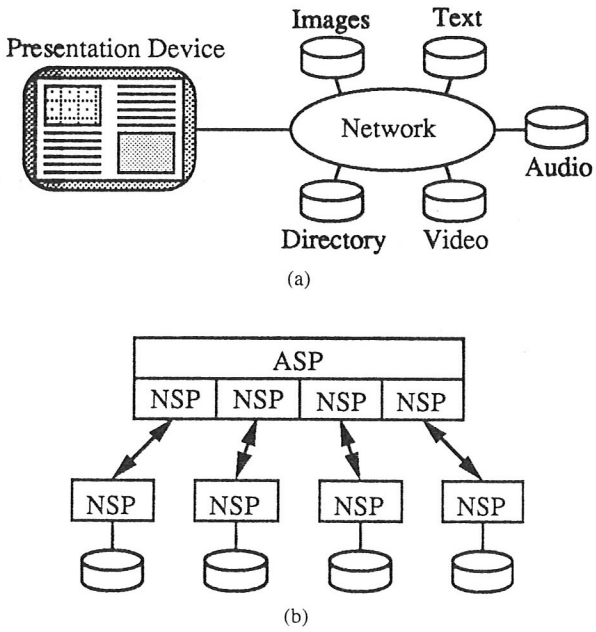

Fig. 16. (a) Physical distribution of data. (b) Protocol interaction.

VI. Conclusion

In this paper, we have proposed an approach to the synchronization of multimedia data traffic with arbitrary temporal relationships. We employed a single model to handle continuous or synthetic synchronization from database storage or live real-time sources. Accordingly, protocols are defined to provide synchronization of data elements with these arbitrary temporal relationships. These protocols, the NSP and ASP, establish a general set of service primitives for the provision of synchronization as a network service for individual connections or for complex requirements of an application's object as specified by a Petri net. The NSP and ASP map to the OSI reference model session and application layers, respectively.

Additional problems for future consideration include the development of protocols for negotiating QOS parameters between the application and the access mechanism, such as delay, bandwidth, and packet loss probability; establishing a suitable threshold control time T_E; providing packet reconstruction; and providing feedback control for buffer under- or overflow, which are not considered here.

Further, in a large distributed application, data can be dispersed at many locations and the playout of a complex object comprised of many data elements can require an equivalently large number of synchronized connections. By performing synchronization at intermediate sites, the number of connections handled by a single site can be reduced.

ACKNOWLEDGMENT

The authors could like to thank the reviewers for their constructive comments.

REFERENCES

[1] T. D. C. Little and A. Ghafoor, "Network considerations for distributed multimedia object composition and communication," *IEEE Network*, vol. 4, no. 6, pp. 32–49, Nov. 1990.

[2] G. Barberis and D. Pazzaglia, "Analysis and optimal design of a packet-voice receiver," *IEEE Trans. Commun.*, vol. COM-28, no. 2, pp. 217–227, Feb. 1980.

[3] G. Barberis, "Buffer sizing of a packet-voice receiver," *IEEE Trans. Commun.*, vol. COM-29, no. 2, pp. 152–156, Feb. 1981.

[4] W. A. Montgomery, "Techniques for packet voice synchronization," *IEEE J. Select. Areas Commun.*, vol. SAC-1, no. 6, pp. 1022–1028, Dec. 1983.

[5] M. De Prycker, "Functional description and analysis of a video transceiver for a broad site local wideband communications system," *Esprit'85: Status Report of Continuing Work*, The Commission of the European Communities, Ed. North-Holland, 1986, pp. 1087–1108.

[6] M. De Prycker, M. Ryckebusch, and P. Barri, "Terminal synchronization in asynchronous networks," in *Proc. ICC'87*, Seattle, WA, June 1987, pp. 800–807.

[7] W. E. Naylor and L. Kleinrock, "Stream traffic communication in packet switched networks: Destination buffering considerations," *IEEE Trans. Commun.*, vol. COM-30, no. 12, pp. 2527–2524, Dec. 1982.

[8] C. Adams, and S. Ades, "Voice experiments in the UNIVERSE project," in *Proc. IEEE ICC'85*, Chicago, IL, 1985, pp. 29.4.1–29.4.9.

[9] S. Ades, R.Want, and R. Calnan, "Protocols for real time voice communication on a packet local network," in *Proc. IEEE INFOCOM'87*, San Francisco, CA, Mar. 1987, pp. 525–530.

[10] J. Ma and I. Gopal, "A blind voice packet synchronization strategy," IBM Res. Rep. RC 13893, July 1988.

[11] M. G. Salmony and D. Sheperd, "Extending OSI to support synchronization required by multimedia applications," IBM ENC Tech. Rep 43.8904, Apr. 1989.

[12] T. D. C. Little and A. Ghafoor, "Synchronization and storage models for multimedia objects," *IEEE J. Select. Areas Commun.*, vol. 8, no. 3, pp. 413–427, Apr. 1990.

[13] J. Postel, G. Finn, A. Katz, and J. Reynolds, "An experimental multimedia mail system," *ACM Trans. Off. Inform. Syst.*, vol. 6, no. 1, pp. 63–81, Jan. 1988.

[14] A. Ghafoor, P. Berra, and R. Chen, "A distributed multimedia database system," in *Proc. Workshop Future Trends Distrib. Comput. Syst. 1990s*, Hong Kong, Sept. 1988, pp. 461–469.

[15] J. Gemmell and S. Christodoulakis, "Principles of delay sensitive multi-media data retrieval," in *Proc. 1st Int. Conf. Multimedia Inform. Syst. '91*, Singapore, Jan. 1991, pp. 147–158.

[16] C. Yu, W. Sun, D. Bitton, Q. Yang, R. Bruno, and J. Tullis, "Efficient placement of audio data on typical disks for real-time applications," *Commun. ACM*, vol. 32, no. 7, pp. 862–871, June 1989.

[17] J. Wells, Q. Yang, and C. Yu, "Placement of audio data on optical disks," in *Proc. 1st Int. Conf. Multimedia Inform. Syst. '91*, Singapore, Jan. 1991, pp. 123–134.

[18] C. Nicolaou, "An architecture for real-time multimedia communication systems," *IEEE J. Select. Areas Commun.*, vol. 8, no. 3, pp. 391–400, Apr. 1990.

[19] *1st Int. Workshop Network Operat. Supp. Digit. Audio Video*, Berkeley, CA, November 1990; also ICSI Tech. Rep TR90-062, 1990.

[20] D. P. Anderson, S. Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews, "Support for continuous media in the dash system," in *Proc. 10th Int. Conf. Distrib. Comput. Syst.*, Paris, France, May 1990, pp. 54–61.

[21] D. P. Anderson, R. G. Herrtwich, and C. Schaefer, "SRP: A resource reservation protocol for guaranteed-performance communication in the internet," ICIS Tech. Rep. TR-90-006, Feb. 1990.

[22] D. P. Anderson, R. Govindan, and G. Homsy, "Abstractions for continuous media in a network window system," in *Proc. 1st Int. Conf. Multimedia Inform, Syst. '91*, Singapore, Jan. 1991, pp. 273–298.

[23] W. H. Leung, T. J. Baumgartner, Y. H. Hwang, M. J. Morgan, and S. C. Tu, "A software architecture for workstations supporting multimedia conferencing in packet switching networks," *IEEE J. Select. Areas Commun.*, vol. 8, no. 3, pp. 380–390, Apr. 1990.

[24] J. S. Sventek, "An architecture for supporting multi-media integration," in *Proc. IEEE Comput. Soc. Off. Automat. Symp.*, Apr. 1987, pp. 46–56.

[25] G. Karlsson and M. Vetterli, "Packet video and its integration into the network architecture," *IEEE J. Select. Areas Comm.*, vol. 7, no. 5, pp. 739–751, June 1989.

[26] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE J.Select. Areas Commun.*, vol. 8, no. 3, pp. 368–379, Apr. 1990.

[27] D. Ferrari, "Client requirements for real-time communication services," *IEEE Commun. Mag.*, vol. 28, no. 11, pp. 65–72, Nov. 1990.

[28] A. A. Lazar, G. Pacifici, and J. S. White, "Real-time traffic measurements on MAGNET II," *IEEE J. Select Areas Commun.*, vol. 8, no. 3, pp. 467–483, Apr. 1990.

[29] S. Mazumdar and A. A. Lazar, "Monitoring integrated networks for performance management," in *Proc. IEEE Int. Conf. Commun.*, Atlanta, GA, Apr. 1990, pp. 289–294.

[30] A. A. Lazar, T. Temple, and R. Gidron, "An architecture for integrated networks that guarantees quality of service," *Int. Digit. Analog Cabled Syst*, vol. 3, no. 2, 1990.

[31] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Trans. Comput.*, vol. C-36, no. 8, pp. 933–940, Aug. 1987.

[32] M. L. Zhang and T. Murata, "Analysis of self-stabilizing clock synchronization by means of stochastic Petri nets," *IEEE Trans. Comput.*, vol. 39, no. 4, pp. 597–604, Apr. 1990.

[33] P. Ramanathan, D. D. Kandlur, and K. G. Shin, "Hardware-assisted software clock synchronization for homogeneous distributed systems," *IEEE Trans. Comput.*, vol. 39, no. 4, pp. 514–524, Apr. 1990.

[34] H. L. Hartmann, "Synchronization techniques for digital networks," *IEEE J. Select. Areas Commun.*, vol. SAC-4, no. 4, pp. 506–513, July 1986.

[35] H. J. Chao and C. A. Johnston, "A packet video system using the dynamic time division multiplexing technique," in *Proc. GLOBECOM'86*, Houston, TX, Dec. 1986, pp. 767–772.

[36] R. Steinmetz, "Synchronization properties in multimedia systems," *IEEE J. Select. Areas Commun.*, vol. 8. no. 3, pp. 401–412, Apr. 1990.

[37] J. Suzuki and M. Taka, "Missing packet recovery techniques for low-bit-rate coded speech," *IEEE J. Select. Areas Commun.*, vol. 7, no. 5, pp. 707–717, June 1989.

[38] S. Mazumdar and A. A. Lazar, "Knowledge-based monitoring of integrated networks," in *Proc. 1st Int. Symp. Integrated Network Manag.*, Boston, MA, May 1989, pp. 235–243.

[39] T. D. C. Little and A. Ghafoor, "Spatio-temporal composition of distributed multimedia objects value-added networks," *IEEE Comput.*, vol. 24, no. 10, pp. 42–50, Oct. 1991.

[40] P. Egloff and A. Scheller, "Multi-media documents in broadband ISDN: Development of a generalized telecommunication model in BERKOM," in *Proc. Workshop Future Trends Distrib. Comput. Syst. 1990s*, Hong Kong, Sept. 1988, pp. 103–110.

Thomas D. C. Little (S'82–M'83–S'87–M'91) received the B.S. degree in biomedical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1983, and the M.S. degree in electrical engineering and the Ph.D. degree in computer engineering from Syracuse University, Syracuse, NY, in 1989 and 1991, respectively.

He is currently an Assistant Professor in the Department of Electrical, Computer and Systems Engineering, Boston University, Boston, MA. Since 1983, he has been involved with applica-

tions for embedded real-time computer systems including telecommunications and remote ocean sensing. Prior to joining Boston University, he was supported by the CASE Center at Syracuse University, investigating problems in the development of a distributed multimedia information system. His research interests include distributed systems, computer networking, multimedia database management, and real-time system design.

Dr. Little is a member of the IEEE Computer Society, the IEEE Communications Society, and the Association for Computing Machinery.

Arif Ghafoor (S'83-M'83-SM'89) received the B.S. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan in 1976, and the M.S., M.Phil, and Ph.D. degrees from Columbia University, New York, NY, in 1977, 1980, and 1985, respectively.

Currently, he is an Associate Professor in the School of Electrical Engineering, Purdue University, West Lafayette, IN. He was formerly with Syracuse University from 1984 to 1990, and is a consultant to many companies, including Bell Labs and General Electric, in the area of telecommunications and distributed systems. His research interests include design and analysis of parallel and distributed systems, and telecommunications.

Dr. Ghafoor is a member of Eta Kappa Nu.