A Network-Based Scheme For Synchronization Of Multimedia Streams

Taieb Znati[†,§]

Robert Simon [†]

Brian Field [*]

[†]Department of Computer Science

and [§]Telecommunications Program (LIS)

University of Pittsburgh

Pittsburgh, PA 15260

(znati, simon, field)@cs.pitt.edu

Abstract

The recent advances in data processing and desktop computing, coupled with the advent of high speed communication networks, paved the way to the development of large-scale multimedia computing and communications systems. The presentation of multimedia information requires synchronization between various streams, to insure that logically related data are presented concurrently at the appropriate times. The main challenge in developing distributed multimedia systems is the development of efficient schemes to achieve temporal and intermedia synchronization.

In this paper, a communication scheme to facilitate synchronization among multiple multimedia streams is presented. The basic tenet of the scheme is that by bounding the end-to-end delay on the coordinated streams and limiting jitter, synchronization can be guaranteed. The fundamental characteristics of the proposed scheme include guaranteed support for on-time delivery of packets, jitter control, efficient network admission control mechanisms, and efficient network load based per node delay assignment.

[*]Author's current address is U S WEST Techonologies, Suite 280, 4001 Discovery Drive, Boulder, CO 80303

1 Introduction

Recent advances in high-speed networking technology have created opportunities for the development of a wide spectrum of sophisticated multimedia applications which generate, integrate, process, store, and distribute time dependent and time independent media. The time dependent media typically consists of voice, audio, video, and animation, while the time independent media consists of text, graphics, or drawings[13]. Furthermore, the time dependent media exchanged by multimedia applications often requires timely and predictable delivery[7]. A group of logically related data streams, such as audio and video in a tele-conferencing application, is referred to as a multimedia *call*.

Unlike conventional applications, multimedia applications have an additional dimension which associates the flow of real-time with the presentation of the various data streams. When delivered separately, each media stream causes a distinctly perceivable effect that persists for a certain duration of time at the destination node. It is, however, the composite effects caused by the simultaneous display of all related streams that determine the successful progress of the application along the temporal axis. Therefore, a major requirement of any system supporting multimedia applications is the need to provide synchronization to maintain the temporal relationships between the various data streams of a multimedia object. These relationships may be implicitly specified, as in the case of different media components recorded simultaneously at their respective sites, or explicitly specified, as in the case of a multimedia document consisting of audio, video, text and numeric data originating from a set of database servers connected by high-speed networks.

The synchronization process can be achieved by segmenting each data stream into a timed sequence of application perceivable data segments, referred to as *synchronization units*. Associated with each synchronization unit is a synchronization *interval*, which is the time it takes for the presenting device to process all of the information contained in one unit. The synchronization interval is the same for all media streams in a single call, even though the rate of data production may be different. The length of the synchronization interval typically depends on the persistence duration of data [4].

In order to specify the temporal association between data streams, the application identifies the synchronization units of the related streams that are to be delivered simultaneously to the destination. These units constitute a *data frame*. A typical example of a data frame combines video and audio data segments generated over a synchronization interval. Another example of a data frame is a picture and accompanying sound which results from clicking on a portion of the text in a hypertext document. Each data frame is associated with a *playout deadline*. The playout deadline is an absolute time by which all the information from each synchronization unit of the streams to be synchronized must be available to the presenting devices.

One possible approach to achieve synchronization among a set of related data streams is to enforce synchronization at the orchestration layer [7, 6] or at the transport layer [10]. Based on the application's tolerance to the variability in network delay, this approach encapsulates recovery techniques to compensate for loss or late delivery of data. The degree of variability in the network delay, referred as *jitter*, is usually modeled by two parameters, namely, the maximum network delay and the average network delay. These parameters are used to determine the maximum rate of change in delay over time. The rate of change is then used to specify a level of guarantee, in the form of a probability, on the delay behavior of the network. In general, however, the stochastic nature of the actual delay behavior in a network with no service guarantees is usually far too complex to be concisely captured in a simple probabilistic guarantee with which the boundary conditions are expected to be enforced by the network [6]. Accordingly, the approach proposed in this paper achieves synchronization by virtue of enforcing tight guarantees on end-to-end delays in the network. Providing reliable bounds for maximum end-to-end delay minimizes the buffering requirements needed to smooth delay at the destination. Furthermore, the provision of strict network delay bounds reduces the complexity of the application in dealing with failure of the network to deliver data on time. More importantly, for applications that require deterministic service, it is not possible to guarantee their performance bounds at the orchestration layer, if the network behavior changes as the load fluctuates. Deterministic applications have stringent performance requirements, in terms of throughput, delay, delay jitter and loss rate, and cannot tolerate service disruptions caused by wide variations of the network delay.

There are at least three basic design requirements for effective network support of multimedia synchronization. The first is allowing the application an easy and flexible way to express performance, timing and synchronization requirements. In particular, these requirements, such as end-to-end delay and synchronization interval, must be specifiable on a connection-by-connection basis as well as by media type [11]. This will ensure higher network utilization, since different connections do not always need the same level of network support. This issue becomes even more important in future networks where pricing may be based upon usage. While the network specification must be easily specifiable by the application, it must also allow effective translation into the underlying network support policy.

The second design requirement is for the network to provide some guaranteed level of performance for multimedia connections. This can be accomplished by a combination of resource reservation, admission control and efficient scheduling. In addition, any well-designed network policy must not underutilize available resources in order to provide this performance.

The third design requirement is the need to effectively utilize global network resources. Connection performance guarantees should be met, without unnecessarily impairing the ability of the network to accept new connections. This vital step is often overlooked by many proposed service models, which typically look at performance support on a single node basis only.

In an attempt to achieve these three basic design requests, this paper describes a network architecture to support the timing and synchronization requirements of multimedia calls. The architecture is based on a network level abstraction called $(\Phi, \Delta)-$stream. An instance of a $(\Phi, \Delta)-$stream is a simplex end-to-end communication stream associated with a multimedia call. The parameter Φ represents the synchronization interval and Δ represents the end-to-end delay specified by that call. A $(\Phi, \Delta)-$stream is an association between the call's quality of service specifications and the network resources required to support these specifications. The main characteristics of the

$(\Phi, \Delta)-$stream model include:

- The ability to handle synchronization of both time dependent and time independent data streams in a uniform manner.
- The flexibility to allow the destination to determine its own playout strategy in a predictable manner, rather than having either the source or the network dictate its playout point as in a predicted service model.
- The ability to allocate delay and channel capacity to meet the application requirements, taking into consideration the current load of each node on the routing path. The approach to compute suitable per node delay values to meet the end-to-end delay requirements of a new data stream considers the relative load placed on a node's processing and buffer resources by the set of currently supported data streams and the relative state of the node with respect to other nodes in the path. The delay assignment scheme attempts to balance the load on the path, in order to prevent the formation of potential bottlenecks which may unnecessarily reduce the ability of the path to accept new data streams.

The rest of the paper proceeds as follows: Section 2 provides a description the $(\Phi, \Delta)-$stream call model to support of multimedia timing and synchronization requirements, and a discussion of synchronization in the $(\Phi, \Delta)-$stream framework. The basic design features of the $(\Phi, \Delta)-$stream architecture, including connection specification and a load-based delay assignment, are discussed in section 3. Section 4 shows the admissibility test and scheduling mechanism used to support $(\Phi, \Delta)-$streams on a per node basis. Section 5 discusses the runtime environment used in support of a $(\Phi, \Delta)-$stream, and section 6 discusses the buffering requirements of the $(\Phi, \Delta)-$stream framework. Section 7 describes the call establishment procedure. Section 8 discusses related work, and section 9 presents some conclusions and future work.

2 Call Model in the $(\Phi, \Delta)-$stream Framework

Applications commonly labeled "multimedia" range from highly interactive, such as multi-participant teleconferencing systems, to client-server type, such as accessing a multimedia database. Regardless of their types, the communication requirements of distributed multimedia connections have performance specifications which include maximum acceptable end-to-end delay, synchronization and reliability.

To address the basic requirements of multimedia synchronization and delay specifications, the $(\Phi, \Delta)-$stream framework provides a call model designed to accommodate a large class of multimedia applications in a flexible manner. Connection level requests in the $(\Phi, \Delta)-$stream architecture are based upon an end-to-end multimedia connection management policy called ADP-Group communication, where call management is done in the context of distributed process groups [12]. The model allows the the specification of the performance parameters on a call by call basis. This approach allows more flexibility in accommodating destinations with different physical characteristics.

2.1 $(\Phi, \Delta)-$stream Call Specification

The request specification is done on a connection by connection basis and directly maps into the $(\Phi, \Delta)-$stream service model. Within the $(\Phi, \Delta)-$stream model each network connection request is based only upon parameters which reflect the application traffic profile and delay characteristics and are easily characterizable by the application. These parameters, which are most relevant to multimedia communication, include the synchronization interval, Φ, the number of packets produced over each synchronization interval, $N(\Phi)$, and the maximum acceptable end-to-end delay from source to destination.

The value of Φ is the time interval between two logical synchronization points, and so is dependent on the application's synchronization method. Within this interval, $N(\Phi)$ represents the number of packets that need to be delivered to the destination before the playout deadline.

The end-to-end delay Δ is the maximum acceptable delay which any packet can suffer. The end-to-end delay reflects the playout time instant by which data within the synchronization interval must be received by the destination to guarantee successful synchronization.

Based upon the network request specification, the $(\Phi, \Delta)-$stream service policy at each node provides a *guaranteed* level of service to each accepted connection. This is done by an admissibility test and a set of run-time support scheduling policies. The admissibility test verifies the feasibility of supporting the connection's traffic rate over the selected routing path, while the scheduling policies ensure that the per node delay requirement is enforced and the traffic rate between adjacent nodes is maintained across the routing path.

The main characteristic of the scheduling model in the $(\Phi, \Delta)-$stream framework is its flexibility to accommodate the requirements of the application dynamically in effort to improve resource utilization. In this model, bursty connections do not have to reserve at their peak rate. The scheduling policy manages the bandwidth in a way such that packets which exceed their reserved rate specification will not meet their deadlines only under certain congestion conditions. Furthermore, through the path establishment procedure, the $(\Phi, \Delta)-$stream model provides a way of increasing global performance and utilization of the network. This is achieved by assigning a connection per node delay based on the current load of each node on the routing path. A load based per node delay assignment guarantees that the connection's performance requirements are met, without unnecessarily impairing the strength of the the network to accept future calls.

2.2 Synchronization in the $(\Phi, \Delta)-$stream Architecture

The process of synchronization in the $(\Phi, \Delta)-$stream architecture uses the temporal characteristics of the data to define a sequence of synchronization intervals along the real-time axis. The duration of a synchronization interval, Φ, is determined by the persistence duration of various data. The information generated by the $(\Phi, \Delta)-$streams of within a synchronization interval constitutes a data frame or *playback unit*. In order to achieve synchronization successfully, the delay caused by the network in delivering data frame should not exceed the end-to-end delay, Δ, specified by the application

Consider a multimedia call $C = \langle \Phi, \Delta, < s_1, s_2, ..., s_N > \rangle$, where s_i is a $(\Phi, \Delta)-$stream characterized by its $N_i(\Phi)$, source and destination. In a given synchronization interval I_k of length Φ, $(\Phi, \Delta)-$stream i actually generates a set of packets $P(i,k)$, such that $|P(i,k)| \leq N_i(\Phi)$. The set $D(k) = \{\bigcup P(i,k), i = 1, 2, ...N\}$ constitutes a data frame. Figure 1 shows a single call with multiple $(\Phi, \Delta)-$streams, sending an audio stream, a video stream and text annotation. The destination combines the set of packets, $P(i,k), i = 1, 2, 3$, generated by $(\Phi, \Delta)-$stream $s_i, i = 1, 2, 3$, in an interval I_k into a data frame $D(k)$.

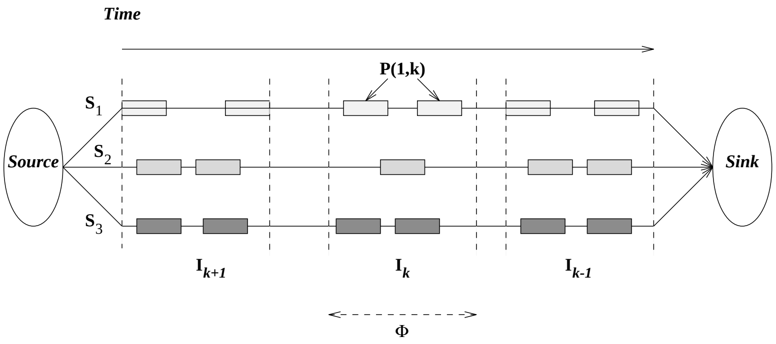

Figure 1: Synchronization Interval for Coordinated Streams

The data frame represents a playback unit presentable at the sink. Based on the above model, synchronization support at the network level consists in delivering all packets of a data frame, $D(k), k = 1, 2, ..$, within the application specified end-to-end delay, Δ. By bounding the end-to-end delay on the coordinated streams and limiting jitter, synchronization can be guaranteed. This network based approach for synchronization support constitutes the basic tenet of the $(\Phi, \Delta)-$stream architecture. The proposed scheme achieves synchronization of related streams based solely on the performance requirements of the applications, without making any assumptions regarding *a priori* knowledge of the statistical behavior of the network.

3 $(\Phi, \Delta)-$stream Network Architecture

The $(\Phi, \Delta)-$stream network architecture supports each call on a per stream basis, rather than a per call basis. Several reasons motivated this decision. Streams may be coming from multiple sources, so they must be supported individually. While each stream in a call has the same value for Δ and Φ, the streams themselves may have different rate specifications. Further, streams within calls may have different performance guarantee needs; some streams may need complete reliability while other streams may be able to suffer a certain amount of packet loss. If streams are bundled together into one connection request, then it may be necessary to distinguish which individual stream a packet belonged to, *in addition* to which connection. This may add an unacceptable amount of overhead.

The $(\Phi, \Delta)-$stream model provides a guaranteed level of performance based upon the call request and the real-time requirements of each stream of the associated multimedia call. The establishment of the $(\Phi, \Delta)-$streams associated with the call and the guaranteed support for real-time performance requirements, including stream synchronization, is a three–part process. The first part is the call connection specification. This involves specifying the destination and the real-time performance requirements of the $(\Phi, \Delta)-$streams associated with the multimedia call.

The second part is the *path establishment* phase. This phase involves selecting a routing path, and specifying the performance requirements for each node on the path. The per node performance specifications are based upon Δ, Φ and $N(\Phi)$, and take into consideration the current node utilization in such a way as to avoid overutilizing any one node while guaranteeing the performance requirements of the multimedia call. During path establishment each node runs an acceptance test to make sure it can meet performance requirements of the new call without violating the guarantees of the currently supported multimedia calls.

Finally, after the $(\Phi, \Delta)-$stream is established, performance guarantees, jitter control and stream synchronization are maintained by a packet traffic shaping mechanism, coupled with a deadline-based scheduling policy using a non-work conserving service discipline. The rest of the paper discusses these issues in detail.

3.1 $(\Phi, \Delta)-$stream Connection Specification

Consider a multimedia call $C = \langle \Phi, \Delta, < s_1, s_2, ..., s_N > \rangle$, where $s_i, i = 1, 2, ...N$ is a $(\Phi, \Delta)-$stream. For each $s_i, i = 1, 2, ...N$, the multimedia call establishment request specifies the following:

$$\langle \Phi, N_i(\Phi), \Delta \rangle$$

where Φ is the synchronization interval, $N_i(\Phi)$ is the maximum number of packets generated by the $(\Phi, \Delta)-$stream over the synchronization interval which require *on-time delivery*, and Δ is the end-to-end delay. A multimedia call is accepted only if all $(\Phi, \Delta)-$streams can be supported by the network.

The ability of an intermediate node to support performance guarantees of a $(\Phi, \Delta)-$stream, consistent with the performance requirements of the underlying multimedia call, depends on the characterization of the resource utilization required at each intermediate node of the routing path. The network run-time support environment must determine whether the nodes on the selected path have sufficient computation and buffer resources to satisfy the requirements of the multimedia call. This process requires the mapping of the rate specification provided by the $(\Phi, \Delta)-$stream multimedia call into a *per node delay specification* and a *per node rate specification.*

Given a request for the establishment of a new multimedia call, the schemes used to compute potential per node delays for each stream and specify a packet rate for each node on the routing path are discussed next.

3.2 Per Node Delay Specification

In order to support performance guarantees, the $(\Phi, \Delta)-$stream framework must provide mechanisms which allow the identification of a set of routing nodes capable of supporting the performance requirements of the multimedia call. Locating a path which connects the source to the destination is a well-known problem, and several algorithms for its solution have been proposed. The details of these algorithms are outside the scope of this paper.

Based on the selected path, a set of delays, the sum of which does not exceed the end-to-end delay specified by the $(\Phi, \Delta)-$stream, is determined for each node on the path. The procedure used to assign possible per node delay values is based on the current utilization of each node on the routing path. The effect is to give more work over the same time interval to an underutilized node rather than an overutilized node on the routing path.

Let $\mathcal{P}$ be a routing path assigned to a new $(\Phi, \Delta)-$stream s. Furthermore, let $\mathcal{N}_A$ and $\mathcal{N}_I$ be the set of active nodes and idle nodes on $\mathcal{P}$, respectively. A node is active if it currently supports at least one $(\Phi, \Delta)-$stream. The main objective of the delay assignment procedure is to utilize the network resources to the largest extent, in order to improve the for the network to perform at a minimal cost. This is achieved based on the metric, ρ_n, which measures the percentage utilization of a node n on the routing path.

Let $S(n)$, the set of $(\Phi, \Delta)-$streams currently supported by an active node n. The percentage utilization of n can be expressed as follows:

$$\rho_n = \sum_{i \in S(n)} \frac{N(\delta_{n,i})}{\delta_{n,i}} \cdot \mu_n \tag{1}$$

where $N(\delta_{n,i})$ is the maximum number of packets generated by $(\Phi, \Delta)-$stream i over the assigned per node delay $\delta_{n,i}$, and μ_n is the service time to transmit a packet at node n. If node n is idle, however, $S(n)$ is empty and ρ_n is set to $\frac{\sum_{j \in \mathcal{N}_A} \rho_j}{|\mathcal{N}_A|}$

The per node delay assignment process selects targeted delay values for each node on the routing path, such that, upon assignment of these delays, the utilization of resources on the routing path remains as balanced as possible. Consequently, if Δ represents the end-to-end delay requirement of $(\Phi, \Delta)-$stream s, and ρ_n represents the percentage utilization of node n, then the per node delay, $\delta_{n,s}$, assigned to node n can be expressed as:

$$\delta_{n,s} = \frac{\rho_n}{\sum_{j \in \mathcal{P}} \rho_j} \cdot \Delta \tag{2}$$

If $\mathcal{N}_A$ is empty, then $\delta_{n,s} = \frac{\Delta}{|\mathcal{P}|}$. A per node delay is called *feasible* if the on-time reliability requirement of the $(\Phi, \Delta)-$stream can be supported by that node over the delay interval.

3.3 Per Node Rate Specification

Consider a $(\Phi, \Delta)-$stream s characterized by its parameters, Φ, $N_s(\Phi)$, and a routing path P. Intermediate nodes on P are labeled $1, 2, ..., |\mathcal{P}|$. Furthermore, let $\delta_{1,s}$ be the per node delay assigned to node 1 for

$(\Phi,\Delta)-$stream s. The maximum number of packets, $N(\delta_{1,s})$, generated by $(\Phi,\Delta)-$stream s over $\delta_{1,s}$ can be expressed as follows:

$$N(\delta_{1,s}) = \left\lceil \frac{\delta_{1,s}}{\Phi} \right\rceil \cdot N_s(\Phi) \tag{3}$$

The expression $\left\lceil \frac{\delta_{1,s}}{\Phi} \right\rceil$, which represents the number of complete intervals of size Φ that fit into $\delta_{1,s}$, is multiplied by $N_s(\Phi)$ to determine the maximum number of packets generated over those intervals. Notice that the translation from the end-to-end specified requirements into the per node requirements does not decrease the specified level of support as required by the application. Consequently, the value $N(\delta_{1,s})$ represents the worst case arrival pattern. In other words, for a given delay value $\delta_{1,s}$, the value $N(\delta_{1,s})$ represents the maximum number of packets node 1 may receive from $(\Phi,\Delta)-$stream s.

The maximum packet arrival rate at an intermediate node $i, i \geq 2$, on the routing path is determined by the maximum number of packets that can be serviced by the previous node $i-1$ within a per node delay $\delta_{i-1,s}$. Consequently, the maximum number of packets that can be received by a routing node $i, i \geq 2$, over a per node delay $\delta_{i,s}$ can be expressed as:

$$N(\delta_{i,s}) = \left\lceil \frac{\delta_{i,s}}{\delta_{i-1,s}} \right\rceil \cdot N(\delta_{i-1,s}), \quad i \geq 2 \tag{4}$$

Based on this rate specification, each node on the routing path determines the feasibility of supporting the on-time reliability requirements of $(\Phi,\Delta)-$stream s, so that no packet misses the $(\Phi,\Delta)-$stream assigned delay. The feasibility test must also verify that the new $(\Phi,\Delta)-$stream can be supported without violating the performance requirements of the currently support $(\Phi,\Delta)-$streams. If the specified rate is feasible, the $(\Phi,\Delta)-$stream framework uses a set of runtime scheduling policies to maintain the traffic rate at the specified levels across the routing path and guarantee the on-time delivery requirements of the supported multimedia call. Section 4 describes the feasibility test performed by each node, while Section 5 discusses the run-time scheduling policies.

4 Per Node Delay Feasibility Test

The main idea underlying the $(\Phi,\Delta)-$stream acceptance process scheme is based on the relationship among the maximum number of packets received by a given node over a given time interval, the per-packet processing time and the delay a packet may suffer at that node. This relationship states that the knowledge of both the maximum number of packets received by a given node over a given time interval and the per-packet processing time determines the maximum delay a packet may suffer at that node. More specifically, if $N(\delta)$ is the maximum number of packets generated over an interval of size δ, and μ is the packet service time then the maximum delay, d, experienced by any packet verifies:

$$if \ \ N(\delta) \cdot \mu \leq \delta, \ \ then \ \ d \leq \delta \tag{5}$$

To determine the feasibility of accepting a new $(\Phi, \Delta)-$stream s, node, n, on the routing path must verify that the above relationship holds in the case where:

- The given time interval δ is the per-node delay, $\delta_{n,s}$, supported by the intermediate node.
- The maximum number of packets $N(\delta)$ is the maximum number of packets, $N(\delta_{n,s})$, that could be generated by the $(\Phi, \Delta)-$stream over $\delta_{n,s}$.
- The packet processing time μ is the time required to process a packet from $(\Phi, \Delta)-$stream s.

The feasibility of multimedia call can be verified by applying the acceptance process to all $(\Phi, \Delta)-$streams. To describe the node verification process at an intermediate n, the details are first introduced by discussing how the algorithm works for a single node supporting a single $(\Phi, \Delta)-$stream.[1] The discussion is further extended to the case of two streams, and later to N streams traversing a single node. The feasibility test assumes that the packet scheduling policy at node n is non-preemptive and that packets from $(\Phi, \Delta)-$stream j are serviced before packets from $(\Phi, \Delta)-$stream k if $\delta_j \leq \delta_k$. A detailed description of the $(\Phi, \Delta)-$stream run-time support mechanisms is provided in section 5.

4.1 Case of a Single Stream

Consider a $(\Phi, \Delta)-$stream, s, routed through node n, and characterized by the maximum number of packets generated over an interval of size δ_s is $N(\delta_s)$. Since the node scheduling policy is non-preemptive, and assuming s is the only $(\Phi, \Delta)-$stream currently supported by the intermediate node, service of the guaranteed packets generated by s may be delayed at most for the time required to service one non-guaranteed packet. This packet may have started receiving service at the time packets from s arrive. The maximum number of packets to be serviced within an interval of size δ_s is, therefore, bound by $N(\delta_s) + 1$.

Based on the above observation, it is sufficient that relation (6) is satisfied for the verification algorithm to provide guaranteed support for the real-time requirement of $(\Phi, \Delta)-$stream s.

$$\mu + N(\delta_s) \cdot \mu < \delta_s \tag{6}$$

4.2 Case of Two Streams

The process described above may be extended to verify the feasibility of supporting a new $(\Phi, \Delta)-$stream, k, without violating the real-time requirements of a currently supported $(\Phi, \Delta)-$stream j. The non-trivial case of the verification algorithm results when the two streams share at least one node, say n, along their routing paths.

[1] For clarity, the subscripts n, denoting an intermediate node on the routing path, will be dropped unless required to prevent ambiguity. Consequently, the per node delay of $(\Phi, \Delta)-$stream s at node n will simply be denoted δ_s.

Let δ_j, $N(\delta_j)$, δ_k and $N(\delta_k)$ be the real-time requirements of $(\Phi, \Delta)-$stream j and k at node n. Assume also, without loss of generality, that $\delta_j < \delta_k$.

The verification process must guarantee that all packets, generated in the worst-case scenario by both streams, are serviced within their delay bounds. From the schedulers perspective, the worst-case arrival pattern results when both streams j and k start sending their maximum number of packets at the same time and the packets of each stream arrive back-to-back.

Based on the scheduling policy, packets from stream j are serviced before packets from stream k. Therefore, the verification process must determine whether the intermediate node has sufficient resources to service the maximum number of packets from stream j that may require service during an interval δ_k, and still met the service requirement of the packets from stream k.

This alignment of packets from stream j and k that produces the worst case scenario during an interval of size δ_k is depicted in Figure 2. In this figure, the intervals $I_1(j,k)$ and $I_2(j,k)$ represent the particular alignment of the δ_j and δ_k intervals that maximizes the number of packets from j that arrive in a single δ_k interval. Consequently, in order to determine whether a given node in the path can support the real-time requirements of both $(\Phi, \Delta)-$streams j and k, the maximum amount of time required to service the packets generated by each $(\Phi, \Delta)-$stream in each interval, must be computed. For each interval, this amount of time is denoted by $I_1^{st}(j,k)$ and $I_2^{st}(j,k)$, respectively.

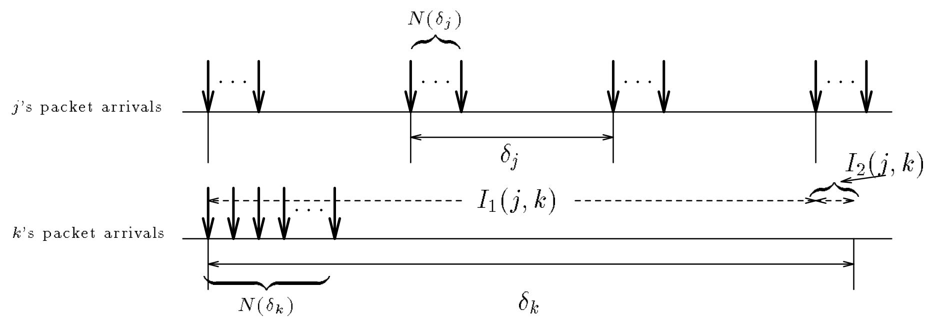

Figure 2: Maximum Number of Packet Alignments for Two $(\Phi, \Delta)-$streams

The size of $I_1(j,k)$ can be determined by computing the maximum number of complete intervals of size δ_j that can fit in an interval of size δ_k. Therefore, the size of $I_1(j,k)$ is computed as follows:

$$I_1(j,k) = \left\lfloor \frac{\delta_k}{\delta_j} \right\rfloor \cdot \delta_j. \tag{7}$$

During each interval of size δ_j in $I_1(j,k)$, the node may need to service $N(\delta_j)$ packets from $(\Phi, \Delta)-$stream j. The time $I_1^{st}(j,k)$ to service packets from j during $I_1(j,k)$ can then be computed as follows:

$$I_1^{st}(j,k) = \left\lfloor \frac{\delta_k}{\delta_j} \right\rfloor \cdot N(\delta_j) \cdot \mu. \tag{8}$$

The interval, $I_2(j,k)$, denotes the portion of δ_k that is not covered by $I_1(j,k)$. The verification process must account for the portion of packets from $(\Phi,\Delta)-$stream j that may require service during this partial interval. Depending on the size of $I_2(j,k)$ the node may need to account for all $N(\delta_j)$ packets generated by j during an interval δ_j, or only consider the portion $\left\lceil \frac{I_2(j,k)}{\mu} \right\rceil$ of these packets. The amount of time required to service these packets is computed as follows:

$$I_2^{st}(j,k) = min(N(\delta_j)\cdot\mu, \left\lceil \frac{I_2(j,k)}{\mu} \right\rceil \cdot \mu) \tag{9}$$

where the function $min(x,y)$ returns the minimum of x and y's values.

The sum of $I_1^{st}(j,k)$ and $I_2^{st}(j,k)$, therefore represents the maximum amount of time the node may spend servicing packets from j over *any* interval of size δ_k. During an interval of size δ_k, the node must also service the packets from k, which contributes $N(\delta_k)\cdot\mu$ units of time. Hence, $(\Phi,\Delta)-$stream k is accepted if the following inequality holds:

$$\mu + I_1^{st}(j,k) + I_2^{st}(j,k) + N(\delta_k)\cdot\mu < \delta_k. \tag{10}$$

If relation (10) is violated, the request of $(\Phi,\Delta)-$stream k is rejected. The function, $\mathcal{A}(j,k)$, used by the verification process to compute the sum of the service time intervals, $I_1^{st}(j,k)$ and $I_2^{st}(j,k)$, is described in Figure 3.

function $\mathcal{A}(j,k)$

$I_1(j,k) = \left\lfloor \frac{\delta_k}{\delta_j} \right\rfloor \cdot \delta_j$

$I_1^{st}(j,k) = \left\lfloor \frac{\delta_k}{\delta_j} \right\rfloor \cdot N(\delta_j)\cdot\mu$

$I_2(j,k) = \delta_k - I_1(j,k)$

$I_2^{st}(j,k) = min(N(\delta_j)\cdot\mu, \left\lceil \frac{I_2(j,k)}{\mu} \right\rceil \cdot \mu)$

return $(I_1^{st}(j,k) + I_2^{st}(j,k))$

end function

Figure 3: Function $\mathcal{A}(j,k)$.

Based on the function $\mathcal{A}(j,k)$, relation (10) reduces to:

$$\mu + \mathcal{A}(j,k) + \mathcal{A}(k,k) < \delta_k. \tag{11}$$

Notice also, that a relation similar to (11) can be easily derived for the single $(\Phi,\Delta)-$stream case by eliminating the term $\mathcal{A}(j,k)$.

Assume instead, that the newly requested $(\Phi, \Delta)-$stream k, has a smaller per node delay than that of the currently accepted $(\Phi, \Delta)-$stream j. In this case, the acceptance scheme must first determine whether $(\Phi, \Delta)-$stream k can be feasibly serviced within its per node delay bounds. This is done by verifying that relation (6) holds for $(\Phi, \Delta)-$stream k. The next step determines whether $(\Phi, \Delta)-$stream j can continue to receive its basic service if $(\Phi, \Delta)-$stream k has been accepted. This is done by verifying that relation (11) holds, with the indices j and k reversed.

In the case where, $\delta_j = \delta_k$, the order in which packets from j and k are serviced is based on the time these packets are inserted into the service queue. Furthermore, the relation (12) holds for any given $(\Phi, \Delta)-$streams j and k which have the same δ value.

$$\mathcal{A}(j,k) + \mathcal{A}(k,k) = \mathcal{A}(k,j) + \mathcal{A}(j,j). \tag{12}$$

As a result, the verification process need only verify that relation (11) holds for it to confirm or deny support for the newly requested $(\Phi, \Delta)-$stream.

4.3 Case of Multiple Streams

In this section, we generalize the verification algorithm so that it can be used to verify the feasibility of supporting N $(\Phi, \Delta)-$streams. The node verification process must determine if a new request for the establishment of an $(\Phi, \Delta)-$stream, k, characterized by its real-time performance specification, δ_k and $N(\delta_k)$, can be supported without violating the real-time requirements of the $N-1$ currently supported $(\Phi, \Delta)-$streams.

Based on the value of δ_k, the set of currently supported $(\Phi, \Delta)-$streams can be divided into two subsets, C_1 and C_2. The set C_1 contains all the $(\Phi, \Delta)-$streams that are characterized by a smaller per node delay than δ_k. Packets from these $(\Phi, \Delta)-$streams are serviced *before* packets from $(\Phi, \Delta)-$stream k. The set C_2 contains all the $(\Phi, \Delta)-$streams that are characterized by larger per node delays than δ_k. Packets from these $(\Phi, \Delta)-$streams can only be serviced *after* packets from $(\Phi, \Delta)-$stream k.

Let S be the set of currently supported $(\Phi, \Delta)-$streams, and assume that $|S| = N-1$. The two sets, C_1 and C_2, may be formally defined as:

$$C_1 = \{b, b \in S \mid \delta_b \le \delta_k\}, \tag{13}$$

$$C_2 = \{a, a \in S \mid \delta_a > \delta_k\}, \tag{14}$$

The verification process to guarantee all $(\Phi, \Delta)-$streams in $S \cup \{k\}$ uses a two phase approach.

In the first phase, the verification algorithm determines whether the requirements of $(\Phi, \Delta)-$stream k can be met, while servicing the maximum number of packets from each $(\Phi, \Delta)-$stream b in C_1. The time required to service all guaranteed packets from these $(\Phi, \Delta)-$streams over an interval of size δ_k is $\sum_{b \in C_1} \mathcal{A}(b,k)$. Furthermore, the verification algorithm must account for the time $\mathcal{A}(k,k)$ required to service all guaranteed packets from

$(\Phi, \Delta)-$stream k. As previously stated, however, service of the packets may be delayed by the time required to service a single packet from non-guaranteed traffic. Therefore, in order to guarantee the support of $(\Phi, \Delta)-$stream k, the algorithm must verify that relation (15) is satisfied.

$$\mu + \sum_{b \in C_1} \mathcal{A}(b, k) + \mathcal{A}(k, k) < \delta_k. \tag{15}$$

If the above relation holds, the verification process proceeds with the second phase of the algorithm; otherwise the request for the establishment of $(\Phi, \Delta)-$stream k is rejected.

The purpose of the second phase to determine that the acceptance of $(\Phi, \Delta)-$stream k does not cause any guaranteed packets from $(\Phi, \Delta)-$streams in C_2 to violate their per node delay bounds. Packets from $(\Phi, \Delta)-$streams in C_2 can only be serviced after the packets from $(\Phi, \Delta)-$stream k. Consequently, in order to guarantee that the real-time requirements of $(\Phi, \Delta)-$streams in C_2 are preserved, the verification algorithm must verify that these requirements hold for every individual $(\Phi, \Delta)-$stream in C_2. More specifically, the algorithm verifies that for each $(\Phi, \Delta)-$stream a in C_2, there is sufficient time to service the maximum number of guaranteed packets from $(\Phi, \Delta)-$stream i, where $\delta_i \leq \delta_a$. This verification procedure continues until all channels in C_2 have been satisfactorily verified. The request for the establishment of the new $(\Phi, \Delta)-$stream is denied if it is determined that an $(\Phi, \Delta)-$stream a in C_2 has its requirements violated by the acceptance of $(\Phi, \Delta)-$stream k. Therefore, the algorithm must support that relation 16 is satisfied.

$$\forall a \in C_2 \quad \mu + \sum_{\forall i \in C_1 \cup C_2 \cup k, \delta_i \leq \delta_a} \mathcal{A}(i, a) + \mathcal{A}(a, a) < \delta_a \tag{16}$$

5 Runtime Support in $(\Phi, \Delta)-$stream Network

In order to provide guaranteed service, the run-time support environment must address the issue of supporting the $(\Phi, \Delta)-$streams on-time reliability requirement. Furthermore, the environment must address the issue of defining a scheduling strategy that guarantees that packets are serviced within their delay bounds. Finally, adequate buffer space must be available at all intermediate nodes on the routing path and at the destination.

5.1 Traffic Classes and Performance Guarantees

Each $(\Phi, \Delta)-$stream associated with a multimedia call is a simplex end-to-end communication $(\Phi, \Delta)-$stream which provides an application with guaranteed performance for the support of timing and synchronization. There are two classes of traffic within a $(\Phi, \Delta)-$stream. The first is referred to as *Basic Traffic* (*BT*). This class denotes all of the information $N(\Phi)$ contained in a synchronization interval Φ. Upon establishment, the $(\Phi, \Delta)-$stream guarantees delivery of at over each time interval Φ at least $N(\Phi)$ packets, and that the worst case delay suffered by any packet is not greater than Δ.

The second class of traffic is *Enhancement Traffic* (ET). This is traffic generated by the application in excess of its rate specification, whose on-time delivery will enhance the basic level of service. The delivery of this traffic depends on the load of the network nodes. When delivered, the ET traffic is delivered with a maximum end-to-end delay of no more than Δ.

5.2 Policing

A policing mechanism is used in the $(\Phi, \Delta)-$stream architecture to ensure that each $(\Phi, \Delta)-$stream abides by its specified traffic requirements at all times. This mechanism is required by the $(\Phi, \Delta)-$stream model to support real-time guarantees, and is done by considering the different classes of traffic to enforce the $(\Phi, \Delta)-$stream's real-time requirements and traffic specifications. The proposed scheme uses packet marking as an integral part of $(\Phi, \Delta)-$stream's specification to control real-time traffic. Marking, however, is not used to deny packets access to the network, but rather to keep track of their rate. This feature allows the scheme to provide the application with flexible, yet controllable, network access.

The packet policing mechanism uses a moving window policy, in order to enforce rate control of real-time traffic [9]. Recent traffic history for each real-time application is used to verify that the application is obeying its rate specification. The policing scheme, however, is optimistic in its approach to deal with excess traffic. The adopted strategy does not delay excess traffic at the application or the network boundary. Instead, the packets are marked appropriately to reflect their type and injected into the network immediately. This scheme is referred to as the *optimistic moving window (OMW)* packet policing strategy.

Each packet of a $(\Phi, \Delta)-$stream is *statically* marked either BT or ET when it enters the network. A packet from a given $(\Phi, \Delta)-$stream is marked BT only if its marking does not cause the basic on-time reliability traffic to exceed its specified rate value. If marking such a packet would cause the BT traffic to exceed its rate specified value, the packet is marked as ET.

Excess packets, ET, are injected into the network *immediately* rather than delaying them at the network boundary. Marking and injecting excess packets immediately into the network preserves their chances of getting delivered on-time. However, these packets can only be serviced if their service does not cause the network to violate the requirement of the supported $(\Phi, \Delta)-$stream. Allowing applications to violate their traffic rate provides more flexibility to highly bursty applications as well as to applications that may have a poor approximation of their exact bandwidth requirements.

The OMW mechanism uses a strategy similar to the user-behavior envelope (UBE) scheme described in [15], but achieves a different purpose. The OMW mechanism is used to implement a two level control strategy that identifies the different traffic classes. It does not, however, aim at constraining the application's generation pattern.

5.2.1 On-Time Reliability Support

The first run-time support design issue addresses the question of how the intermediate nodes should cooperate to support the end-to-end reliability requirement of an $(\Phi, \Delta)-$stream. The run-time environment must determine the percentage of an application's $(\Phi, \Delta)-$stream packets traversing through an intermediate node that need to be serviced on-time so that the application's end-to-end reliability is met.

In a framework where all intermediate nodes operate independently, the lack of run-time communication between the nodes may cause the level of support required by each node to be significantly larger than the required end-to-end on-time reliability. This may unnecessarily constrain the node resources.

In a more cooperative framework, nodes on the routing path may provide dynamic on-time reliability support based on the level of support provided by other nodes. It is not clear, however, how effective this scheme would be in a network supporting a variety of applications with widely varying traffic loads and real-time performance requirements. The solution adapted in our scheme relies entirely on the explicit marking performed at the network boundary by the network policing function.

The marking strategy enforced at the network boundary allows the intermediate nodes to identify different types of traffic and support the $(\Phi, \Delta)-$stream's basic traffic on-time reliability requirement. More specifically, for a given $(\Phi, \Delta)-$stream s, characterized by its on-time reliability and its rate specification interval, the network has to guarantee that at most $N(\delta_{1,s})$ are marked as BT from among the packets generated by $(\Phi, \Delta)-$stream s. This control is enforced over any interval of size $\delta_{1,s}$ by marking a packet from $(\Phi, \Delta)-$stream s as being BT only if, during the immediately preceding interval of size $\delta_{1,s}$, no more than $N(\delta_{1,s}) - 1$ other packets were marked as BT. If marking the packet as BT would exceed the specified rate over $\delta_{1,s}$, the packet would then be marked as ET. This scheme allows lightly loaded intermediate nodes to utilize their excess processing capacities to enhance the service provided to the $(\Phi, \Delta)-$streams, while guaranteeing their minimum end-to-end on-time reliability throughout the routing path.

5.2.2 $(\Phi, \Delta)-$stream Scheduling Policies

The second design issue of the run-time support addresses the requirements of the packet scheduling strategy. The goal of this strategy is to maintain the traffic rate at the specified level across the network and guarantee that the on-time reliability requirements of the $(\Phi, \Delta)-$streams are supported. This goal is achieved by two scheduling policies. The first policy preserves the per node traffic rate specification by regulating the traffic between intermediate nodes. The second policy aims at servicing packets from $(\Phi, \Delta)-$streams within their per node delay bounds.

5.2.3 Traffic shaping and jitter control

As traffic travels across a routing path, clustering of packets may occur at the intermediate nodes. This accumulation of packets distorts the initial shape of the traffic and causes per node delay variations. The amount of distortion that may occur depends on many factors, including the number of nodes traversed, the intermediate node scheduling policies, and the relative processing speed differences among nodes of the routing path. In order to compensate for the delay variations introduced by previous nodes, each network node on the routing path maintains a *waiting room* for each $(\Phi, \Delta)-$stream. The waiting room is used to absorb the jitter introduced upon a packets from a $(\Phi, \Delta)-$stream at the previous node of the routing path.

When a $(\Phi, \Delta)-$stream s is established a per node a delay $\delta_{n,s}$ is assigned to each intermediate node n on the routing path. A packet is considered *early* at n if the actual time, $\hat{\delta}_{n-1,s}$, spent at a node $n-1$ is less than $\delta_{n-1,s}$. The *earliness* of a packet, $\delta_{n-1,s} - \hat{\delta}_{n-1,s}$, determines the amount of time an early packet from s is held in n's waiting room before it is considered eligible for service and moved to its appropriate service queue.

The jitter control scheme is a non-work conserving mechanism which does not require any quantitative or qualitative knowledge of the $(\Phi, \Delta)-$stream's specified traffic profile. The architecture makes no explicit or implicit assumptions about the traffic pattern or profile. By absorbing the jitter, the traffic pattern of a $(\Phi, \Delta)-$stream can effectively be reconstructed at each intermediate node to the form it has when it entered the network.

5.2.4 Service Queues

To support servicing packets within their per node delay bounds, the intermediate nodes maintain two service queues. Each service queue holds packets from a specific traffic class. The BT service queue holds packets that are eligible to be serviced, and are marked as BT. Similarly, the ET queues hold eligible ET packets. These packets are ordered in their respective queues based on their per node delay bounds.

Each service queue is assigned a service priority. The BT queue has priority over the ET queue. The scheduler provides a non-preemptive service to all packets in the BT queue. Packets with the smallest per node delay bounds are serviced first. Ties among eligible packets are broken based on the time these packets were inserted into the service queue. When the BT queue becomes empty, the scheduler moves to service any packets in the ET queue. Packets in this queue are also serviced based on their delay bounds. However, ET packets that exceed their per node delay are dropped.

Notice that the policing mechanism used by the $(\Phi, \Delta)-$stream framework may result in packets from a given application arriving out of sequence. These packets contain sufficient information to indicate their relative position within the original traffic stream. Furthermore, these packets are guaranteed to be delivered within their end-to-end delay bound. This allows the application to perform any required synchronization of these packets.

The scheduling strategy described above guarantees that packets are serviced within their per node delay bounds, given that the intermediate node has reserved sufficient resources to service these packets. The verification scheme used by the intermediate nodes determines the amount of resources necessary to preserve the basic requirements of all supported $(\Phi, \Delta)-$streams. The description of this scheme is the focus of the next section.

6 Buffer Allocation Requirements

Buffer requirements arise from two different parts of the $(\Phi, \Delta)-$stream model. The first is the per stream requirements, based on the service policy described above, and the second is at the destination.

6.1 $(\Phi, \Delta)-$stream Buffer Requirements

Based on the packet scheduling strategy the number of buffers required for each $(\Phi, \Delta)-$stream in each intermediate node can be directly deduced from the number of packets the $(\Phi, \Delta)-$stream can generate over a per node delay δ. A packet from an $(\Phi, \Delta)-$stream can be delayed either at the waiting room or at the service queue.

The eligibility mechanism may cause a packet to be delayed in the waiting room of the intermediate node. The delay, however, does not exceed $\delta - \mu$. During this time, at most a total of $N(\delta) - 1$ other packets may also be present in the queue. Once the packet becomes eligible and enters the service queue, the packet may be further delayed for at most $\delta - \mu$ amount of time. During this time interval, up to $N(\delta) - 1$ other packets may become eligible and placed into one of the service queues. Therefore, the number of buffers required by a $(\Phi, \Delta)-$stream in each node is at most $2 \cdot N(\delta)$. Notice that, this number considers all expected packets. Therefore, the number of buffers required by an $(\Phi, \Delta)-$stream at an intermediate node is at most $2 \cdot N(\delta)$.

Based on the above analysis, the verification algorithm not only verifies that the real-time requirements of the newly requested and currently supported $(\Phi, \Delta)-$streams are preserved, but also that $2 \cdot N(\delta)$ buffers are available to support traffic from the new $(\Phi, \Delta)-$stream. The $(\Phi, \Delta)-$stream request for establishment is rejected if either of the above requirements is not met.

6.2 Receiver Buffer Requirements

As described in section 7 each $(\Phi, \Delta)-$stream in a call may be routed on a different path between source and destination. To account for different actual end-to-end delays the outcome of the path establishment procedure must inform each sink how much buffering may required for each connection, which may be more than the amount of space needed for one synchronization interval. This is done as follows.

Assume that a call has k $(\Phi, \Delta)-$streams. Then for each of the k paths which are assigned to these streams we may rank the actual end-to-end delays on each path as $\Delta_1 \leq \Delta_2 \leq \ldots \leq \Delta_{l-1} \leq \Delta_k \leq \Delta$. Note that there is no requirement that these paths are disjoint. Let b_i be the maximum number of packets which the destination must be able to buffer from $(\Phi, \Delta)-$stream i for each synchronization playback unit. If $(\Phi, \Delta)-$stream i assigned to a path P^i, with an associated delay Δ_i, then

$$b_i = \left(\left\lceil \frac{\Delta_k - \Delta_i}{\Phi} \right\rceil + 1 \right) \cdot N(\Phi)$$

Based on b_i the destination needs to provide buffer space for each stream which is part of the call. Notice that this space is fixed throughout the lifetime of the call.

7 $(\Phi, \Delta)-$stream Call Establishment Procedure

Assume that node e, at the network edge, receives a request for an establishment of a call C. The request specifies a destination node d, an end-to-end delay Δ, a synchronization interval Φ, and K $(\Phi, \Delta)-$streams, each of which is characterized by $N_s(\Phi), s = 1, 2, ...K$. In order to verify the feasibility of accepting this call and guaranteeing the performance requirements of its associated $(\Phi, \Delta)-$streams, the following steps are performed :

1. Node e produces K $(\Phi, \Delta)-$stream requests of the form $\langle \Phi, N_s(\Phi), \Delta \rangle$, $s = 1, 2, ...K$.
2. Node e forms each per node $(\Phi, \Delta)-$stream delay request:
   - Select K routing paths $P_1, \ldots, P_K$ from e to d. Notice that these paths are not necessarily disjoint.
   - For each path P_s, selected for $(\Phi, \Delta)-$stream s, determine the vector of the per-node delay values, $\delta_{n,s}$, for each node, n, on the routing path.
3. Attempt to establish the path:
   - For each node $n \in P_s$ determine the per node rate specification $[N(\delta_{n,s}), \delta_{n,s}]$ based on equation 3.
   - At each node $n \in P_s$ run feasibility test described in section 4. If this test is passed then forward the $(\Phi, \Delta)-$stream establishment request to next node on path, else reject the request.
   - If all nodes $n \in P_s$ mark the $(\Phi, \Delta)-$stream as feasible, then mark $(\Phi, \Delta)-$stream request s as accepted, else mark it as rejected.
4. If all paths P_s mark all $(\Phi, \Delta)-$streams as accepted then the call is accepted else either reject or lower the quality of service requirements of one or more $(\Phi, \Delta)-$streams.

Node s performs step 2 on the basis of its most recent knowledge of the status of the network. The alternate is to perform all of step 2 on a hop by hop basis. The path establishment procedure overcomes failure of the initial attempt to establish a call by allowing the application to re-negotiate its quality of services to an acceptable level of service. This feature provides more flexibility in specifying the quality of service requirements and results into accepting more calls.

8 Related Work

There have been many proposed models for supporting multimedia synchronization at the network level, including Hierarchical Round Robin (HRR) [5], Jitter Earliest Due Date [14], Stop-and-Go Queueing [3], and the α-channel [2]. On a per node basis the $(\Phi, \Delta)-$stream scheme bears similarities to some of the proposed schemes, but differs from them in a variety of ways.

The $(\Phi, \Delta)-$stream framework uses the application's end-to-end delay specification, synchronization interval and strength of the path overall to determine the appropriate δ value over which to meet the required end-to-end delay. Consequently, the $(\Phi, \Delta)-$stream provides a more flexible approach to accommodate the real-time requirements of the supported applications. Furthermore, the end-to-end delay is guaranteed regardless of the routing path traversed by the $(\Phi, \Delta)-$stream.

Other models for end-to-end synchronization control, such as [1], and [8] have also been proposed. In [1] a scheme is proposed for a flow synchronization protocol which can be applied to multimedia applications. The scheme assumes globally synchronized clocks and adapts to changing network delays. In [8], the proposed scheme does not assume the existence of globally synchronized clocks. Instead, synchronization is achieved by adaptive media-specific feedback techniques for synchronous retrieval from multimedia on demand services. This occurs in the presence of network delay jitter, and nondeterministic playback rate mismatches.

The $(\Phi, \Delta)-$stream model has several features which distinguishes it from the above models. Each call's connection specification includes only parameters which can be easily specified by the application , thereby avoiding complicated or inefficient mappings from connection specification to network request. By controlling jitter and delay the application or the network need make no assumption about the selected routing paths. Further, unlike many of the other node service models described above, the $(\Phi, \Delta)-$stream model has an efficient method for assigning per node delays in a real-time systems service architecture based on the node utilization.

9 Conclusions

We have presented the $(\Phi, \Delta)-$stream model as a framework for providing synchronization support for a distributed multimedia system. The fundamental mechanism by which this is done is through controlling end-to-end delay and jitter on communication streams which are to be synchronized. The $(\Phi, \Delta)-$stream architecture provides a flexible connection specification mechanism to account for the synchronization requirements of the underlying multimedia application. Each $(\Phi, \Delta)-$stream is provided with an appropriate level of network support by an acceptance test, which ensures that the performance of a new $(\Phi, \Delta)-$stream can be supported while maintaining the levels of performance for currently supported $(\Phi, \Delta)-$streams. By taking into consideration the relative utilization of a node on the routing path, the $(\Phi, \Delta)-$stream model assigns per node delays for real-time connections which maximizes the strength of the network while providing guaranteed levels of support for each new and previously accepted call.

Future work includes research into routing and path establishment issues, as well as supporting synchronization in a non reliable network environment.

References

[1] Escobar, J., Partridge, C., and Deutsch, D., "Flow Synchronization Protocol," *IEEE/ACM Transactions on Networking*, Vol. 2, No. 2, April 1994, pp. 111-121.

[2] Field, B., and Znati, T. "α-Channel: A Network Framework to Support Real-Time Performance Guarantees," *IEEE JSAC*, Vol. 11, No. 8, August 1993

[3] Golestani, S.J., "A Framing Strategy for Congestion Management," *IEEE JSAC*, Vol. 9., No. 7, September 1991

[4] Herrtwich, R.G., "Timed Data Streams in Continuous Media Systems," in TR90-017, I.C.S.I., Berkeley, CA. May 1990.

[5] Kalmanek C.R., Kanakia H., and Keshav S., "Rate Controlled Servers for Very High-Speed Networks," *GlobeCom 90*, 1990, pp. 12-20.

[6] Little, T.D.C., and Ghafoor, A., "Multimedia Synchronization Protocols for Broadband Integrated Services," *IEEE JSAC*, Vol. 9, No. 9, December 1991, pp. 1368-1381.

[7] Nicolaou, C., "An Architecture for Real-Time Multimedia Communication Systems," *IEEE JSAC*, Vol. 8. No. 3, April 1990, pp. 391-400.

[8] Ramanathan, S., and Rangan, P.V., "Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks," *IEEE/ACM Trans. on Networking*, Vol. 1, No. 2, pp. 246-260, April 1993

[9] Rathgeb, E.P., "Modeling and Performance Comparison of Policing Mechanisms for ATM Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 2, pp. 325-334, April 1991.

[10] Ravindran K., and Bansal V., "Delay Compensation Protocols for Synchronization of Multimedia Data Streams," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, pp. 574-589, August 1993.

[11] Simon, R., Sclabassi, R., and Znati, T., "Communication Control in Computer Supported Cooperative Work Systems," In Proceedings of *ACM CSCW '94*, Raleigh, N.C., October 24-26, 1994, pp. 311-322.

[12] Simon, R., Znati, T., and Sclabassi, R., "Group Communication in Distributed Multimedia Systems", in *14th International Conference on Distributed Computing Systems*, June 21-24, 1994, Poznan, Poland, pp. 294-303.

[13] Steinmetz, R., "Synchronization Properties in Multimedia Systems", *IEEE JSAC*, Vol. 8, No. 3, April 1990, pp. 401-411.

[14] Verma, D., Zhang, H., and Ferrari, D., "Delay jitter control for real-time communication in a packet switching network," in *Proc. TriComm '91*, 1991, pp. 35-43

[15] Zhang, L., "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks", *SIGCOMM '90*, 1990, pp. 19-29.