Parallel Bindings in Distributed Multimedia Systems

Michael Repplinger*, Florian Winter*, Marco Lohse* and Philipp Slusallek*
*Computer Graphics Lab, Department of Computer Science,
Saarland University, 66123 Saarbrücken, Germany
Email repplinger@cs.uni-sb.de, fw@graphics.cs.uni-sb.de, mlohse@cs.uni-sb.de, slusallek@cs.uni-sb.de

Abstract— **Today's multimedia environments are characterized by different stationary and a growing number of mobile devices, like PDAs or even mobile phones. To meet their requirements regarding configuration and adaptation, multimedia middleware systems often use a flexible flow graph based approach in which multimedia data is processed while flowing from sources to sinks. Here the application has full control about all components and can reconfigure the entire flow graph if needed, which requires sending arbitrary reliable control information in addition to the multimedia data.**

This results in a single multimedia data stream consisting of ordered messages with different Quality of Service (QoS) requirements in terms of reliability and deadline. To meet all these requirements, different types of messages must be transmitted using different transport protocols, while the ordering constraints between messages must still be preserved. For this purpose we propose the concept of a *parallel binding* which allows for the usage of multiple transport protocols to send messages of a single stream with mixed QoS requirements. We also present synchronization algorithms for multiplexing messages after transmission and reconstruction of the original message order of the stream.

I. Introduction

The diversity of stationary and mobile devices of today's multimedia environments results in highly dynamic and heterogeneous systems. Therefore, a maximum of configurability and adaptivity of the underlying software is needed to meet the specific requirements of applications and media in addition to the timing requirements of multimedia data. Traditional streaming services or middleware systems like CORBA [1] provide only limited access to and configuration of their underlying components, and thus are not able to meet all of these requirements.

Although some CORBA implementations like *The Ace Orb* (TAO) [2] support explicit binding and predictable realtime QoS, they do not support automatic runtime reconfiguration and adaptation [3]. In contrast, today's distributed multimedia middleware systems like TOAST [4], Infopipes [5] or the Network Integrated Multimedia-Middleware (NMM) [6] are especially designed for multimedia processing. These multimedia platforms support *explicit binding* to handle application and media specific requirements of transporting multimedia data streams in networks. A *binding object*, introduced by the RM-ODP standard [7], represents a connection between two components as first class object to the application and allows for specifying the desired transport protocol and its specific parameters according to the requirements of transported multimedia data. Infopipes uses the concept of pipes to connect processing components and uses netpipes for data transport. NMM uses a flexible flow graph based approach in which processing components are represented as nodes and multimedia data flows along the edges from sources to sinks. In contrast to traditional streaming clients, the application or any middleware service is able to control each component or reconfigure the entire flow graph during runtime, if needed.

Such high configurability and adaptivity permits changes of a single component, which may affect the remaining components as well. Especially changes related to the data stream, such as a new image resolution of a processed video stream due to adaptivity reasons, requires updating all participating components. Another example is the need for resynchronization when timestamps of a data stream are no longer continuous, which for example occurs during channel switching on a digital TV receiver. This information must be embedded directly within the multimedia data stream. Thus, it is insufficient to only send multimedia data between components, but it is vital to also send arbitrary control information for updating all participating components of a flow graph or informing the application about changes related to the multimedia data stream itself.

Additionally, sending arbitrary information between the components can be used to inform the application about events, like the progress of a task or about the beginning or the end of a media track or file. We found that at least three different kinds of messages of a multimedia data stream must be supported by a flow graph based multimedia middleware for realizing distributed multimedia applications.

1) *Multimedia buffers* carry realtime multimedia data in a specific data format between the components of a flow graph.
2) *Reliable events* carry arbitrary critical information between the components of a flow graph, such as changes of image resolution or the beginning of a new track.
3) *Unreliable events* carry arbitrary non-critical or frequent information, like the progress of a task.

A typical multimedia stream in a flow graph based multimedia middleware may consist of all of these kinds of messages, that are transported between the nodes through binding objects, as seen in Figure 1. Multimedia buffers with certain realtime constraints typically have the highest density in a stream, whereas reliable events appear only infrequently at critical points. Since reliable events usually carry mandatory information about the data stream, it is vital that no reliable events are lost along the

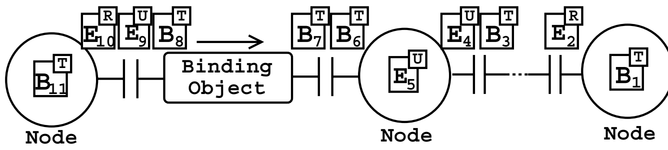

Fig. 1. A typical multimedia stream in a flow graph based middleware consists of different kinds of ordered messages with mixed Quality of Service (QoS) requirements in terms of reliability and deadlines. Multimedia Buffer B are used to carry multimedia data with timing constraints T. Reliable events E^R are used to carry arbitrary critical information, whereas unreliable events E^U are used to carry arbitrary non-critical or frequent information. The messages are processed by the nodes and sent to the successor node through a binding object.

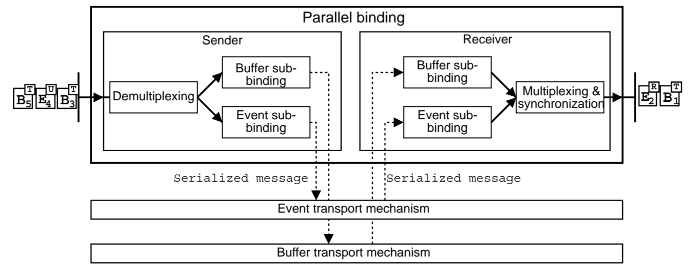

Fig. 2. This parallel binding consists of two sub-bindings, one for Buffer and for Events. At the sender side, incoming reliable events E^R and unreliable events E^U are forwarded to the Event sub-binding and incoming multimedia Buffers B with timing constraints T to the Buffer sub-binding. Each sub-binding uses its specific transport mechanism to send the messages to the receiver. At the receiver side the multimedia data stream is multiplexed again based on the sequence numbers of messages and passed on to the receiving component.

edges of the flow graph. Multimedia buffers, on the other hand, have certain realtime constraints, especially in live multimedia streams.

In contrast to common streaming applications where a weak order of control information is sufficient, a distributed multimedia application may require a strict order of different kinds of messages in a stream. For example if a video stream changes resolution, and it contains a control event that informs all processing components about the resolution change, then all buffers in the stream after the event must contain data of the new resolution. In this case it is vital for all components that the ordering constraints of messages are preserved by the binding objects for correct processing by the nodes. The result is a single ordered multimedia data stream consisting of different kinds of messages with different Quality of Service (QoS) requirements in terms of reliability and deadline.

Common transport protocols can not be used for sending a multimedia data stream with mixed QoS requirements across a single network connection, because they are not able to meet all requirements of these different message types. Reliable transport protocols, such as TCP, which can transport arbitrary information can be used for sending mandatory control events, but can not fulfill the timing requirements of multimedia data. Reliable message transport requires acknowledgments and retransmission to ensure that all messages have been received. This disagrees with the requirements of multimedia data where it is acceptable to sacrifice reliability in favor of meeting the timing requirements of already received messages. Unreliable transport protocols that can transport arbitrary information, such as UDP are unsuitable for sending mandatory events. Furthermore, like reliable protocols, they don't take the timing requirements of multimedia data into account. To meet the specific requirements of multimedia data, user-level protocols like the *realtime transport protocol* (RTP) [8] were proposed. For example, RTP is based on UDP and extends it with flow control, congestion control and timing functionality for multimedia data. Unfortunately, for this purpose, additional knowledge of the data format is required, so that these protocols can only transport multimedia data of certain formats, which also makes them unsuitable for transmitting arbitrary control events.

Thus, new transport protocols like the *Stream Control Transmission Protocol* (SCTP) or the transport protocol for *heterogenous packet flows* (HPF) were proposed. The SCTP protocol can transport multiple independent streams with different QoS properties in terms of reliability and order of messages within a single connection [9]. This is especially useful for media streams that are partially ordered rather than strictly ordered. Unfortunately, it doesn't provide re-ordering of messages from independent streams to a single stream again. In contrast, the HPF protocol can be used to transport data with mixed QoS properties in terms of reliability, priority, and deadline across the same transport connection [10]. Therefore, it meets the requirements of multimedia data streams with arbitrary control information. Although support for such powerful protocols can easily be added to multimedia middleware systems, they are not very popular yet and their availability can not be assumed in distributed and highly heterogenous network environments.

A completely orthogonal approach is the use of a combination of multiple network connections using different transport protocols to transport messages with different QoS requirements. However, since the ordering constraints of the stream must be preserved, messages must be resynchronized after reception. Especially the synchronization task is too complex to be left to the application programmer. Therefore, we argue that a multimedia middleware must support the use of such a combination of multiple transport protocols at the middleware layer and propose the concept of *parallel binding*.

First, we will introduce the concept of parallel binding with its requirements in Section II. Section III presents synchronization strategies which are required to recreate the original message order of a data stream at the receiver side. In Section IV we will present and discuss our implementation and performance measurements. Finally, Section V concludes this paper.

II. The concept of parallel binding

As already mentioned in Section I, binding objects are used to represent a connection between two components as first class objects to the application. A single transport protocol

is used by each binding object. Since explicit binding only enables the configuration and control of the used transport protocol, different binding types have already been proposed. An *open binding* allows for specifying an object graph, that consists of network components representing a network connection as well as processing objects like QoS monitors or rate control components [11]. Furthermore, it provides interfaces to access and control these components. In addition, an open binding allows for the dynamic reconfiguration of its internal object graph, which enables the application to add (or remove) for example additional transcoding objects to reduce bandwidth requirements if needed. But the open binding is limited to the usage of a single binding object for the network connection and doesn't support multiple parallel network connections.

This concept was extended to a *hierarchical binding* [12], which provides a hierarchical structure to create a binding object from a high-level description, e.g. a QoS specification provided by the application. The specification is then mapped onto a hierarchy of multimedia processing components and respective binding objects. Each binding object is responsible for allocation and configuration of its binding components or mapping onto further binding objects. Even this approach is limited to a single binding object for each network connection and doesn't allow the usage of different network connections for transporting the messages of a single data stream.

Therefore we propose the concept of *parallel binding*. The basic idea of this approach is to extend the concept of a common binding object and to specify and use a set of binding objects, which are called *sub-bindings*, to transmit different kinds of messages of a single multimedia data stream across the network. As seen in Figure 2, incoming messages are demultiplexed and forwarded to the sub-binding which meets their specific QoS requirements. At the receiver side, messages are multiplexed and finally forwarded to the *receiving component*. The application is able to access and control each sub-binding. Based on the available transport protocols, we focus on the following types of sub-bindings which are supported by a parallel binding:

1) *Reliable sub-bindings* use reliable transport protocols for transmitting arbitrary information and thus can be used to send mandatory control events.
2) *Unreliable sub-bindings* use unreliable transport protocols for transmitting arbitrary information and can be used for sending non-mandatory control events.
3) *Multimedia sub-bindings* use unreliable transport protocols which can only transport multimedia buffers of a certain format.

While transport mechanisms are in general treated as "black-box" within a sub-binding, several requirements have to be fulfilled by a sub-binding of a parallel binding.

1) Messages are transmitted in-order through a sub-binding. This means that a message m_j will never arrive through a sub-binding that has received a message m_i with $i > j$.
2) The used transport protocol of a binding, even of a multimedia binding, must be able to transport unique and continuous sequence numbers.
3) If the used transport protocol of a sub-binding changes the sequence numbers of given messages, then the sub-binding must provide a feedback mechanism which informs the parallel binding about the sequence numbers actually used.

Protocols for handling multimedia data streams, like RTP, typically fulfill the first requirement. Also, the second property can be fulfilled by protocols such as RTP that require splitting or merging of media buffers into packets that get new sequence numbers assigned. The third requirement greatly simplifies the multiplexing step. However, it might be not available for a certain transport protocol, but it is no real limitation because a sub-binding can be extended by a component for message ordering.

Notice that for a multimedia sub-binding, we cannot specify which sequence numbers are to be used. In particular, sequence numbers are only unique and continuous for the sub-binding itself, but not for all parallel sub-bindings. Therefore, when using such a multimedia sub-binding in parallel with a sub-binding that allows for arbitrary data to be transmitted, the above defined requirements need to be modified. In particular, a sub-binding that allows for arbitrary data to be transmitted only need to provide monotonic increasing sequence numbers, i.e. a single sequence number can be used for transmitting several message. However, since a multimedia sub-binding uses strictly monotonic increasing sequence numbers that cannot be influenced, the lastly used sequence number of the multimedia sub-binding is being applied for all following messages sent via the other sub-binding. Since this sub-binding allows to transmit arbitrary information, an additional internal sequence number can be used for realizing in-order transmission for the sub-binding itself. Unfortunately this approach can not be used for an arbitrary number of multimedia sub-binding, but allows to treat a single multimedia sub-binding as "black-box".

III. Message Synchronization

In Section I we defined different types of sub-bindings that we distinguish by their QoS properties and their ability to transport arbitrary data. We distinguish between reliable sub-bindings (bindings that can reliably transport arbitrary data), unreliable sub-bindings (bindings that can transport arbitrary data and permit loss of data) and multimedia sub-bindings (bindings that can only transport multimedia buffers and also permit loss of data). We also stated that a parallel binding may consist of any combination of these different kinds of sub-bindings and that messages must be synchronized during multiplexing at the receiver side to preserve the ordering constraints of the stream.

In the following section we will discuss algorithms for message synchronization at the receiver side of parallel bindings with different combinations of the various types of sub-bindings. To illustrate the general idea of message synchronization, we first examine the case of multiple reliable

sub-bindings (Section III-A). Then we extend this algorithm for one reliable and one unreliable sub-binding (Section III-B). After this, we consider multiple unreliable sub-bindings (Section III-C) as an intermediate step before we present the synchronization algorithm for unreliable sub-bindings and a multimedia sub-binding (Section III-D). Next we will focus on combinations of reliable sub-bindings and a multimedia sub-binding (Section III-E). Finally we derive from the previous cases the general situation of multiple reliable and unreliable sub-bindings and a multimedia sub-binding (Section III-F).

A. Synchronization of multiple reliable sub-bindings

To illustrate the general idea, we first examine the most simple case of multiple reliable sub-bindings, which will be used as a basis for the remaining algorithms. In the case that all sub-bindings are reliable, message multiplexing is a straightforward sorting process at the receiver side. Consider a parallel binding with multiple reliable sub-bindings. The basic idea is that messages must be forwarded to the receiving component in-order. So if any sub-binding has received message m_i, it must wait until message m_{i-1} has been forwarded before it can forward message m_i. Since all sub-bindings transmit messages in-order, the sub-binding can assume that a missing message m_j with $j < i$ will arrive from one of the other sub-bindings if it has not been forwarded yet. Reliability of the sub-bindings allows us to wait for any missing message m_j until it arrives. This algorithm requires no queueing of messages at the middleware layer. At most one message must be stored per sub-binding while the sub-binding is waiting for messages to arrive on one of the other sub-bindings.

B. Synchronization of one reliable and one unreliable sub-binding

In this section we will extend the previously discussed synchronization algorithm to support a single reliable and a single unreliable sub-binding. If sub-bindings are unreliable, messages can be lost, which would cause the algorithm to wait infinitely long for a missing message. Sine data loss can only occur at a single unreliable sub-binding, only a minor extension is required.

Let i be the sequence number of the latest message that has been forwarded from the parallel binding to the receiving component. If the next message m_j to be handled has a sequence number $j > i+1$, it cannot be forwarded immediately. Instead, we must determine for all missing messages m_k with $i < k < j$ if it will arrive and therefore we need to wait for it or, if it was lost and we can ignore it. The derived algorithm works as follows. If m_j was received from the reliable sub-binding, all missing messages m_k with $i < k < j$ can only be received from the unreliable sub-binding because in-order reception is provided by all sub-bindings. Since missing messages are transmitted using the unreliable sub-binding, some m_k might get lost. Therefore, the forwarding of m_j is delayed until a message m_l with $l \geq j-1$ was received from the unreliable sub-binding. If this happens, we can conclude that none of the missing messages m_k with $i < k < j$ will be received and forwarded anymore. If m_j was received from the unreliable sub-binding, one of the following two statements is true for each of the missing messages m_k with $i < k < j$.

1) The missing message m_k was sent via the unreliable sub-binding and was lost. This is because m_j with $j > k$ was received from the same unreliable sub-binding and in-order reception is provided by all sub-bindings.
2) The missing message m_k was sent via the reliable sub-binding, will be received, and needs to be forwarded before m_j.

To determine, which of the two cases is true, we extend the information transmitted for each message sent via the unreliable sub-binding and include the sequence number r of the latest reliably sent message. This is possible since an unreliable sub-binding allows to include arbitrary additional information. For the message m_j received from the unreliable sub-binding, the algorithm can determine if one or more reliably sent messages are still to arrive. This is the case if $r > i$. Then, the forwarding of m_j is delayed until message m_r was handled.

Notice that for case 1, the worst case occurs if the last unreliably sent message before a large number of reliably sent messages gets lost. In this case, the forwarding of all reliably received messages is delayed until the next message is successfully received from the unreliable sub-binding. However, control events that are typically sent using a reliable sub-binding only occur very rarely within the stream of multimedia buffers sent over an unreliable sub-binding. Therefore, also for case 2, an additional delay is only introduced if we have to wait for a reliably sent message m_r.

C. Synchronization of multiple unreliable sub-bindings

Before we present a synchronization algorithm for a combination of multimedia sub-bindings, we first consider the combination of multiple unreliable sub-bindings as an intermediate step. The algorithm presented in the previous section still depends on the reliability of one sub-binding. If all sub-bindings are unreliable, data loss can occur at any unreliable sub-binding and we must extend this algorithm.

Again, we keep track of the latest sequence number i that has been forwarded to the receiving component. If a sub-binding receives a message m_j with $j > i+1$, then we have to distinguish two cases for all missing messages m_k with $i < k < j$:

1) Message m_k was lost during transmission by one of the sub-bindings and will never arrive.
2) Message m_k and m_j were sent through different sub-bindings and m_k might still arrive.

Unfortunately, the receiver can not tell on which sub-binding a missing message m_k will arrive and whether it will arrive at all. The solution to this problem is to introduce a timeout for each message m_k after which the message is assumed to be lost. If the message arrives after the timeout has expired, it is no longer guaranteed that it can be forwarded in-order, so we typically have to drop it. This is acceptable because all sub-bindings are unreliable and data loss is permitted.

Thus, the algorithm is as follows. If a sub-binding receives message m_i, it waits until message m_{i-1} has been received and forwarded to the receiving component, or the timeout for message m_{i-1} has expired.

What remains is the difficult task of computing a suitable timeout for each message m_i at the receiver side. If the timeout is too large, the receiver will wait too long, which causes unnecessarily long delays and violates the realtime conditions of a multimedia data stream. If the timeout is too small, too many messages will be dropped, causing unnecessary loss of data. We find that a suitable value for $timeout_i$ is the estimated time when message m_i is expected to be received. This can be estimated as the time t_i^S at which message m_i was sent plus the average transmission delay Δt_i. This allows us to treat a sub-binding as „black-box" and we can simply ignore its details like queueing or splitting of messages. So we can estimate the transmission delay Δt_i for each message m_i based on the time t_i^S it is sent and the time t_i^R it is received. Since we are discussing unreliable sub-bindings that can transmit arbitrary data, we can simply include the value t_i^S into message m_i before sending it and compute the transmission delay Δt_i at the receiver side using the following formula:

$$\Delta t_i = t_i^R - t_i^S \tag{1}$$

For each sub-binding Sub we compute the *average transmission delay* ΔT_{Sub} using an exponential smoothing algorithm, as used in many transport protocols, such as TCP [13]. In contrast to the real average transmission delay, an exponential smoothing algorithm considers the current network conditions much better.

$$\Delta T_{Sub} = \alpha * \Delta t_i + (1 - \alpha) * \Delta T_{Sub}, \ \alpha \in [0;1] \tag{2}$$

Since the sender and receiver hosts have two different clocks, this value includes the offset between the clocks if they are not synchronized. But we use this value to estimate the time of arrival of a message at the receiver side, which must include this clock offset as well. So no synchronization between the sender and receiver clocks is required. If we recompute the average transmission delay at regular intervals, we also take the drift of the two clocks into account.

If we have a message m_{i+1} and we want to compute the timeout for message m_i, we still don't know the time t_i^S when the message m_i was sent. Thus, we include this value into message m_{i+1} before sending it and can use the following formula to compute the timeout for message m_i:

$$timeout_i = t_i^S + \Delta T_{Sub} \tag{3}$$

It is trivial to extend this algorithm from two sub-bindings to n sub-bindings, provided that all sub-bindings are unreliable.

If we have not received any messages yet (and thus have no samples of the transmission delay), we should use a suitable initial value for ΔT_{Sub}. Since we have no samples for Δt_i yet we simply initialize ΔT_{Sub} with a reasonable constant value in our implementation. In the worst case, this leads to either a short delay or data loss at the beginning of the stream until a transmission delay can be computed from the first received message.

D. Synchronization of multiple unreliable sub-bindings and one multimedia sub-binding

Based on the synchronization algorithm presented in the previous Section, we can now simply derive the algorithm for multiple unreliable sub-bindings and a multimedia sub-binding. Since we can not include any additional information into messages sent through a multimedia sub-binding, we don't know the time t_i^S when a message m_i was sent which is required to calculate the average transmission time ΔT_{Sub} and the $timeout_i$ using formula 2 and 3. To measure the average transmission delay for each unreliable sub-binding, we simply include the required information into the messages, as described in Section III-C. But to measure the average transmission delay for the multimedia sub-binding, we need correct sample values for t_i^S and t_i^R at the receiver side. The only way to achieve this is to send the sending times t_i^S from the sender to the receiver. This can be done by generating middleware internal sender reports. These reports can be transmitted over one of the unreliable sub-bindings. If only a multimedia sub-binding is used, we need an additional internal control binding for sending the reports. Fortunately, in a typical scenario we have at least one sub-binding which is able to transmit arbitrary data.

To compute the timeout for a missing message m_i that has been received from a multimedia sub-binding using formula 3 we immediately need its sending time t_i^S and can not wait until the corresponding sender report is available. However, we can assume that message m_i has been sent through the sub-binding Sub some time before message m_{i+1} was sent through sub-binding Sub'. Thus, we can use its sending time to compute the timeout as follows:

$$timeout_i = t_{i+1}^S + \Delta T_{Sub} \tag{4}$$

The sending time t_{i+1}^S of a received message m_{i+1} from sub-binding Sub' can be estimated as follows:

$$t_{i+1}^S = t_{i+1}^R - \Delta T_{Sub'} \tag{5}$$

Unfortunately, if the difference between t_{i+1}^S and t_i^S is very large, the approximation is insufficient and the receiver waits too long, which violates the realtime conditions of the multimedia data stream. However, if messages are sent at a fixed rate through a sub-binding Sub, its average sending interval could be used to improve the approximation.

Especially multimedia buffers of a live stream or certain unreliable events, e.g. progress events of a task are sent with an average gap Δt_g between the messages. It can be computed as the average over the intervals between the sending times t_k^S of the messages m_k with $k = 0, \ldots, i-1$.

$$\Delta t_g = \frac{\sum_{k=0}^{i-1} t_{k+1}^S - t_k^S}{i} \tag{6}$$

The sending time t_{i+1}^S of message m_{i+1} can then be estimated as

$$t_{i+1}^S = t_{i+1}^R - \Delta T_{Sub'} - \Delta t_g \tag{7}$$

But in general the receiver can not assume an average gap between messages because control events are transmitted at irregular intervals. In this case we have to use the estimated sending time t^S_{i+1} of the received message m_{i+1} as approximation.

Now we can replace the estimated sending time t^S_{i+1} for a missing message m_i in formula 3 with our new approximation and get the following formula if an average gap between received messages can be assumed for sub-binding Sub:

$$timeout_i = \underbrace{t^R_{i+1} - \Delta T_{Sub'} - \Delta t_g}_{\text{estimated sending time } t^S_i} + \Delta T_{Sub} \tag{8}$$

If an average gap can not be assumed, we use the following general formula:

$$timeout_i = \underbrace{t^R_{i+1} - \Delta T_{Sub'}}_{\text{estimated sending time } t^S_i} + \Delta T_{Sub} \tag{9}$$

E. Synchronization of multiple reliable sub-bindings and one multimedia sub-binding

When using multiple reliable sub-bindings and one multimedia sub-binding in parallel, the algorithm described above cannot be applied because we must ensure that all reliable messages are forwarded in order to the receiving component. So we can not use a timeout to wait for a missing reliable message. Therefore, in this section we will present a different solution that is suitable for synchronizing reliable and a multimedia sub-binding.

Again, let i be the latest sequence number that has been forwarded to the receiving component, let m_j be the pending message and let m_k with $i < k < j$ be the missing messages that have not yet been received. We first consider the case of one reliable and one multimedia sub-binding. If m_j has been received from the reliable sub-binding, we can again (as discussed in Section III-B) assume that all missing messages m_k are unreliable. However, if message m_j has been received from the multimedia sub-binding, we have two possible cases for each missing message m_k:

1) The message m_k is reliable and will be received from the reliable sub-binding.
2) The message m_k is unreliable and has been lost during transmission over the multimedia sub-binding.

In the scenario of Section III-B, the receiver could distinguish between these two cases using the sequence number of the previous reliable message m_r that was stored in message m_j. As already mentioned, it is not possible to insert arbitrary data into multimedia buffers. Since message m_j may be the only message that has arrived at the receiver so far, the receiver has no way to tell whether it has to wait for any missing reliable messages or not. Although the sender has no way to tell the receiver whether a stream contains a reliable message before a multimedia buffer m_j, it is able to simply avoid this situation, as follows: Whenever a reliable message m_j has been sent, the sender stops sending further messages until m_j has been received. This requires an acknowledgment from the receiver which can be sent upstream through the reliable sub-binding. An obvious drawback of this approach is that the transmission of the whole stream is delayed by one round trip. Fortunately, this request-response scheme is only necessary for mandatory events. As already discussed in Section I, a typical multimedia stream only contains a small number of reliable events. Therefore the delay introduced by our solution is acceptable in most common cases.

To extend this algorithm to an arbitrary number of reliable sub-bindings, we simply store the sequence number of the preceding reliable message m_r in each reliable message and use it to decide whether any reliable message is missing if m_j has been received from an reliable sub-binding. For the case that m_j has been received through a multimedia sub-binding, no extensions are necessary.

Furthermore, we identified our approach to be the only possible solution that allows to consider multimedia sub-bindings as “black-box” as described above. The only facility that needs to be provided is a feedback mechanism that allows to determine the lastly used sequence number for sending messages via the multimedia sub-binding, as already discussed in Section II.

F. Synchronization of arbitrary sub-bindings

Finally, based on synchronization algorithms presented in Sections III-A to III-E, we can now easily derive a synchronization algorithm for one reliable, one unreliable and one multimedia sub-binding that can be used for any number of reliable, unreliable sub-bindings and one multimedia sub-binding as well.

The general algorithm is essentially the one presented in Section III-E. The presence of unreliable sub-bindings that can transport arbitrary data doesn’t add any additional problems, because a unreliable sub-binding can be treated in the same way as a multimedia sub-binding. The presence of a multimedia sub-binding forces the sender to wait for acknowledgments for reliable events, as discussed in Section III-E. Of course this is only necessary, if multimedia buffers are sent after reliable events. However, this can usually be assumed since a multimedia stream consists mainly of multimedia buffers.

It is preferable for a multimedia middleware to provide all of the algorithms discussed in the previous sections and always use the most special one to provide the best possible performance.

IV. Implementation and Performance Measurements

We implemented the concept of parallel binding using the Network-Integrated Multimedia Middleware (NMM) [6]. It allows for plugging in different networking components, which are represented by so called *transport strategies*. Therefore the parallel binding is realized as transport strategy using the composite design pattern [14]. Thus, the so called *composite strategy* can be used like any other transport strategy, and available transport strategies can be used as sub-bindings and added according to the requirements of the multimedia data

stream. In addition to this, the composite strategy provides methods to access the used transport strategies and their specific interfaces.

Our composite strategy provides two synchronization mechanisms for different combinations of the various kinds of sub-bindings, which includes two of the most common cases of two reliable and one reliable and one multimedia sub-binding, for which we provide benchmark results in this paper. The first case allows for example the usage of two reliable transport protocols such as TCP with possibly differently optimized network parameters for sending different types of messages. In the second case, one reliable transport protocol like TCP can be used to transport reliable control events while the multimedia buffers are transmitted through a special multimedia protocol such as RTP.

To measure the performance of the synchronization algorithms presented in this paper, we use a simple benchmark application which transmits a stream of control events and buffers of constant size containing random data across the network. Using this application, we run three benchmarks with different configurations:

- Benchmark I uses a single TCP network binding to transmit both buffers and events. This benchmark serves as a reference. No parallel binding is used here.
- Benchmark II uses a composite strategy with two TCP sub-bindings and the synchronization algorithm for two reliable sub-bindings as presented in Section III-A. Control events are sent through the first sub-binding and buffers are sent through the second sub-binding.
- Benchmark III also uses a composite strategy with two TCP sub-bindings, but the synchronization algorithm for one reliable and one multimedia sub-binding is used. Control events are sent through the "reliable" sub-binding and buffers are sent through the "multimedia" sub-binding. Simulating a multimedia sub-binding using TCP permits us to compare the overhead of the parallel binding for different synchronization strategies and a fixed combination of sub-bindings.

The data stream consists of buffers of a constant size of 2048 bytes and infrequent control events. For each configuration, we run the benchmark with a varying number of control events per buffer and measure the average throughput in bytes per second. This allows us to visualize the impact of message synchronization on the overall network performance, depending on the frequency of control events in the data stream. The results of our benchmarks are presented in Figure 3.

As can be seen, the synchronization algorithms add only a small overhead to network performance, provided that the number of mandatory control events is reasonably small. Since a typical multimedia data stream is unlikely to contain more than one event every 100 buffers (which is the best case in our benchmarks), the results are quite acceptable. Events which have to be sent more frequently are candidates for unreliable transmission and thus don't affect performance as much as mandatory events that have to be transmitted

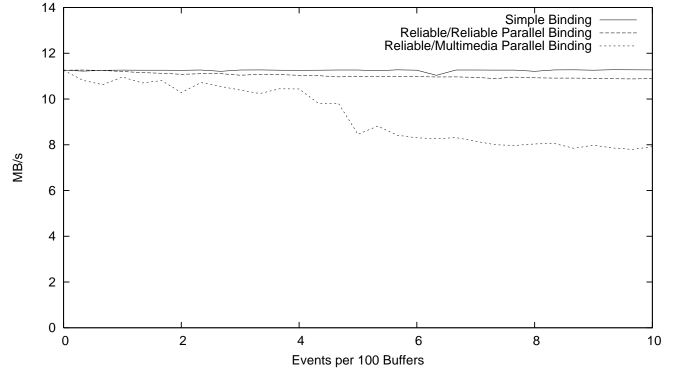

Fig. 3. Throughput benchmark results for a parallel binding using two different synchronization strategies compared with a simple TCP binding

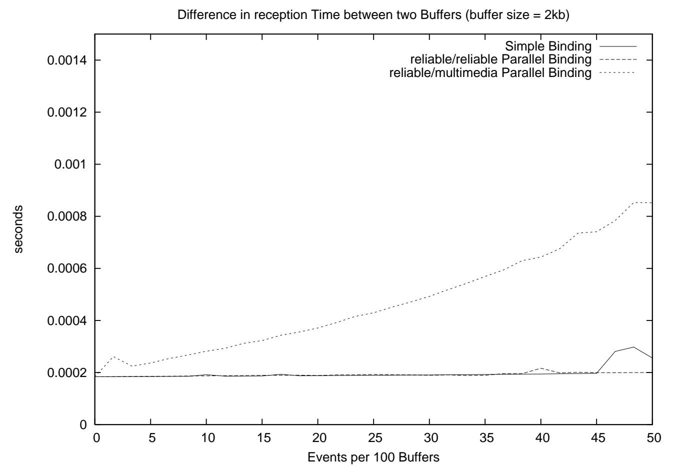

Fig. 4. Average difference in reception time between a buffer and its previous buffer, depending on the density of events and the used synchronization strategy

reliably. Figure 4 shows the average difference in reception time between a buffer and its previous buffer depending on the density of events and the used synchronization strategy is shown. This value only depends on the density of events for the synchronization algorithm for reliable and multimedia bindings. The remaining synchronization algorithms are not affected because no additional acknowledge must be sent after receiving an event and the average reception time between two buffers is nearly constant.

To measure the impact of this acknowledge and the introduced additional jitter in more detail, we performed another benchmark that uses the same configuration as Benchmark III. Again, we run the benchmark with a varying number of control events per buffer and measure the difference in reception time between a buffer and its previous buffer. This allows us to visualize the additional introduced jitter which is caused by the acknowledgement that must be sent after receiving an event. The results of this benchmark are presented in Figure 3. As can be seen, the jitter between buffers without events is about

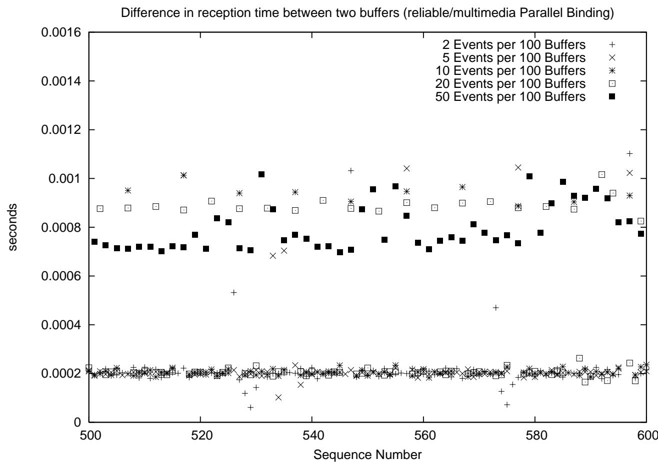

Fig. 5. Difference in reception time between a buffer and its previous buffer, depending on the density of events and using a parallel binding with a reliable and a multimedia sub-binding. As can be seen, events introduce additional jitter due to the acknowledgement mechanism

$0.2\ ms$ in our environment. Transmitting events between two buffers introduces an additional jitter of about $0.8\ ms$, which is even acceptable for most high quality video transmissions, such as a DVD data stream.

V. Conclusions

In this paper we showed that a flow graph based multimedia middleware must support sending arbitrary control information in addition to the multimedia data. Therefore we argued that a multimedia middleware must support the usage of a combination of different transport protocols at the middleware layer for transporting messages with mixed QoS requirements of a single multimedia stream and proposed the concept of *parallel binding* as a solution. Furthermore, we presented different synchronization algorithms to recreate the original message order at the receiver according to the ordering constraints of the messages. Finally we discussed the implementation of a parallel binding and presented some performance measurements which have shown that the performance overhead caused by the synchronization algorithms is acceptable in a typical scenario.

References

[1] Object Managment Group, *Common Object Request Broker Architecture: Core Specification – Version 3.0.3*. OMG, 2004.

[2] I. Pyarali, D. C. Schmidt, and R. Cytron, “Achieving end-to-end predictability of the tao real-time corba orb,” in *Proceedings of the 8th IEEE Real-Time Technology and Applications Symposium*, 2002.

[3] F. Eliassen, A. Andersen, G. S. Blair, F. Costa, G. Coulson, V. Goebel, ivind Hansen, T. Kristensen, T. Plagemann, H. O. Rafaelsen, K. B. Saikoski, and W. Yu, “Next generation middleware: Requirements, architecture, and prototypes,” in *Proceedings of 7th Workshop on Future Trends of Distributed Computing Systems (FTDCS’99)*. Cape Town, South-Africa: IEEE, Dec. 1999.

[4] T. Fitzpatrick, J. J. Gallop, G. S. Blair, C. Cooper, G. Coulson, D. A. Duce, and I. J. Johnson, “Design and Application of TOAST: An Adaptive Distributed Multimedia Middleware,” in *Interactive Distributed Multimedia Systems, 8th International Workshop, IDMS 2001, Proceedings*, 2001.

[5] A. P. Black, J. Huang, R. Koster, J. Walpole, and C. Pu, “Infopipes: an abstraction for multimedia streaming,” *Multimedia Syst.*, vol. 8, no. 5, pp. 406–419, 2002.

[6] M. Lohse, M. Repplinger, and P. Slusallek, “An Open Middleware Architecture for Network-Integrated Multimedia,” in *Protocols and Systems for Interactive Distributed Multimedia Systems, Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems, IDMS/PROMS 2002, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2515. Springer, 2002, pp. 327–338.

[7] Gordon Blair and Jean-Bernard Stefani, *Open Distributed Processing and Mulitimedia*. Addison-Wesley, 1998.

[8] H. Schulzrinne, S. Casner, R. Frederick, , and V. Jacobson, “RFC 1889: RTP: A Transport Protocol for Real-Time Applications,” http://www.ietf.org/rfc/rfc1889.txt, 1996.

[9] L. Coene, “RFC 3257: Stream Control Transmission Protocol Applicability Statement,” http://www.ietf.org/rfc/rfc3257.txt, 2002.

[10] J.-R. Li, S. Ha, and V. Bharghavan, “HPF: A Transport Protocol for Heterogeneous Packet Flows in the Internet,” in *Proceedings of IEEE Infoeom*, 1999, pp. 543–550.

[11] T. Fitzpatrick, G. Blair, G. Coulson, N. Davies, and P. Robin, “Supporting Adaptive Multimedia Applications through Open Bindings,” in *Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDS ’98)*, 1998.

[12] D. Waddington and G. Coulson, “A Distributed Multimedia Component Architecture,” in *Proceedings of 1st International Enterprise Distributed Object Computing Conference (EDOC ’97)*, 1997.

[13] D. D. Clark, “RFC 813: Window and Acknowledgement Strategy in TCP,” http://www.ietf.org/rfc/rfc813.txt, 1982.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.