CORRELATED MULTI-STREAMING IN DISTRIBUTED INTERACTIVE MULTIMEDIA SYSTEMS

BY

POOJA AGARWAL

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Professor Klara Nahrstedt
ABSTRACT

Distributed Interactive Multimedia Environments (DIMEs) enable geographically distributed people to interact with each other in a joint media-rich virtual environment for a wide range of activities, such as art performance, medical consultation, sport training, etc. The real-time collaboration is made possible by exchanging a set of multi-modal sensory streams over the network in real time. The characterization and evaluation of such multi-stream interactive environments is challenging because the traditional Quality of Service metrics (e.g., delay, jitter) are limited to a per stream basis. In this work, we present a novel ”Bundle of Streams” concept to define correlated multi-streams in DIMEs and present new cyber-physical, spatio-temporal QoS metrics to measure QoS over bundle of streams. We realize Bundle of Streams concept by presenting a novel paradigm of Bundle Streaming as a Service (SAS). We propose and develop SAS Kernel, a generic, distributed, modular and highly flexible streaming kernel realizing SAS concept. We validate the Bundle of Streams model by comparing the QoS performance of bundle of streams over different transport protocols in a 3D tele-immersive testbed. Also, further experiments demonstrate that the SAS Kernel incurs low overhead in delay, CPU, and bandwidth demands.
To my parents, for their love and support.
ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Klara Nahrstedt for her guidance and support and for nurturing a keen research attitude in me through the freedom of thought. This work couldn’t have been possible without her invaluable advice and feedback in times of need. I acknowledge the research grants NSF CNS 09-64081, NSF CNS 08-34480, NSF CNS 07-20702, and NSF CNS 10-12194 on which this research is funded. I am thankful to my parents; my mother and father for their always available love and support even from far-away land, India. I am thankful to all the Monet members for the helpful discussions we had in the lab, and for their excellent help during grueling times of paper submissions and other deadlines.

I am thankful to Almighty God, for giving me right opportunities to pursue my dreams. Last but not the least, I am thankful to Rishi Agarwal, my best friend who’s constant encouragement and moral support have been the driving factors of my career. I dedicate all my success to him.
# TABLE OF CONTENTS

LIST OF TABLES ......................................................... vii

LIST OF FIGURES ...................................................... viii

LIST OF ABBREVIATIONS ............................................... ix

CHAPTER 1  INTRODUCTION ........................................... 1
  1.1 Problem Description ........................................... 2
  1.2 Contributions .................................................. 2
  1.3 Outline ......................................................... 3

CHAPTER 2  RELATED WORK ........................................ 4
  2.1 DIME Overview ................................................ 4
  2.2 DIME Systems: TEEVE ....................................... 6
  2.3 QoS Metrics for Synchronous Data Delivery ................. 9
  2.4 Streaming Service Provision ................................ 10

CHAPTER 3  BUNDLE OF STREAMS ................................ 12
  3.1 Background ..................................................... 12
  3.2 Cyber-Physical Mapping of Sensors to Correlated Streams 13
  3.3 Bundle of Streams Concept .................................. 16
  3.4 Bundle of Bundles ............................................ 18
  3.5 Operations on Bundles ....................................... 18
  3.6 Macroframes ................................................... 19

CHAPTER 4  BUNDLE OF STREAMS QOS METRICS ................. 22
  4.1 Background ..................................................... 22
  4.2 Cyber-Physical Non-Weighted QoS Metrics .................. 23
  4.3 Cyber-Physical Weighted QoS Metrics ....................... 26

CHAPTER 5  SAS KERNEL: BUNDLE STREAMING AS A SERVICE KERNEL FOR CORRELATED MULTI-STREAMING ................. 29
  5.1 Background ..................................................... 29
  5.2 Bundle Streaming as a Service (SAS) Paradigm ............. 30
  5.3 SAS Kernel Framework ....................................... 31
5.4 Stream Flow in SAS Kernel .................................................. 34
5.5 SAS Kernel Design ................................................................. 36

CHAPTER 6 IMPLEMENTATION AND EVALUATION RESULTS .......... 43
  6.1 Evaluation of Bundle of Streams Concept in DIMEs ....................... 43
  6.2 SAS Kernel Implementation Details ........................................... 51
  6.3 Evaluation of SAS Kernel ....................................................... 54

CHAPTER 7 CONCLUSION ............................................................. 58

REFERENCES ................................................................. 60
LIST OF TABLES

3.1 Table of Notations for Bundle of Streams. ........................................ 21
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td></td>
<td>DIME Activities</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td></td>
<td>DIME Definition</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td></td>
<td>A Typical DIME Setup with Sensors</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td></td>
<td>DIME Setup</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td></td>
<td>TEEVE Architecture</td>
<td>7</td>
</tr>
<tr>
<td>2.6</td>
<td></td>
<td>TEEVE 3 Sites Environment</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td></td>
<td>DIME Environment</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td></td>
<td>Mapping from Physical Space to Cyber Space</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td></td>
<td>Cyber-Physical Correlations</td>
<td>15</td>
</tr>
<tr>
<td>3.4</td>
<td></td>
<td>Representation of Bundle of Streams and Bundle of Bundles in a DIME Architecture</td>
<td>17</td>
</tr>
<tr>
<td>3.5</td>
<td></td>
<td>Representation of a Macroframe in a DIME Architecture</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td></td>
<td>Bundle Metric Evaluation Framework</td>
<td>24</td>
</tr>
<tr>
<td>5.1</td>
<td></td>
<td>SAS Kernel Framework</td>
<td>33</td>
</tr>
<tr>
<td>5.2</td>
<td></td>
<td>Distributed SAS Kernel Components</td>
<td>35</td>
</tr>
<tr>
<td>5.3</td>
<td></td>
<td>SAS Kernel Data, Control, and Monitoring Planes</td>
<td>36</td>
</tr>
<tr>
<td>5.4</td>
<td></td>
<td>Socket Interface and Tunneling</td>
<td>37</td>
</tr>
<tr>
<td>5.5</td>
<td></td>
<td>Device Stream Specification for a Video Stream</td>
<td>39</td>
</tr>
<tr>
<td>5.6</td>
<td></td>
<td>S-RTP Header Specification</td>
<td>40</td>
</tr>
<tr>
<td>5.7</td>
<td></td>
<td>SAS Kernel Function Manager</td>
<td>42</td>
</tr>
<tr>
<td>6.1</td>
<td></td>
<td>Experimental Setup with 3 Remote Sites for Bundle of Streams Evaluation</td>
<td>44</td>
</tr>
<tr>
<td>6.2</td>
<td></td>
<td>Comparison of Cyber-Physical Metrics against Traditional Metrics</td>
<td>45</td>
</tr>
<tr>
<td>6.3</td>
<td></td>
<td>Performance of TCP, DCCP, and UDP over Bundle Delay</td>
<td>47</td>
</tr>
<tr>
<td>6.4</td>
<td></td>
<td>Performance of TCP, DCCP, and UDP over Bundle Jitter</td>
<td>47</td>
</tr>
<tr>
<td>6.5</td>
<td></td>
<td>Performance of TCP, DCCP, and UDP over Bundle Skew</td>
<td>48</td>
</tr>
<tr>
<td>6.6</td>
<td></td>
<td>Performance of TCP, DCCP, and UDP over Bundle Throughput</td>
<td>48</td>
</tr>
<tr>
<td>6.7</td>
<td></td>
<td>Performance of TCP, DCCP, and UDP over Weighted Bundle Delay</td>
<td>49</td>
</tr>
<tr>
<td>6.8</td>
<td></td>
<td>Performance of TCP, DCCP, and UDP over Weighted Bundle Jitter</td>
<td>50</td>
</tr>
<tr>
<td>6.9</td>
<td></td>
<td>Performance of TCP, DCCP, and UDP over Weighted Bundle Skew</td>
<td>50</td>
</tr>
<tr>
<td>6.10</td>
<td></td>
<td>SAS Kernel Testbed Setup</td>
<td>52</td>
</tr>
<tr>
<td>6.11</td>
<td></td>
<td>SAS Kernel Evaluation</td>
<td>56</td>
</tr>
</tbody>
</table>
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DTI</td>
<td>3D Tele-immersion</td>
</tr>
<tr>
<td>DIME</td>
<td>Distributed Interactive Multimedia Systems</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>SAS</td>
<td>Streaming as a Service</td>
</tr>
<tr>
<td>TEEVE</td>
<td>Teleimmersive Environment for EVerybody</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Distributed Interactive Multimedia Environments (DIMEs) allow real-time collaborative activities among multiple, geographically distributed sites. DIMEs have been proven useful for many cyberphysical activities such as physical therapy, sport activities, and entertainment. Some of the specific real applications include Teleimmersive Dancing (TED) [1], remote training [2], virtual world gaming [3], and Tai Chi learning.

DIMEs are usually comprised of several sensing and actuating devices (e.g. cameras, microphones, body sensors, displays, haptic devices). At each site, sensing devices capture periodically current activities in DIME, forming a logical snapshot of the data produced by cameras, microphones, and body sensors. In general DIME model, each sensing device produces a different stream of data representing one dimension of the multimedia environment. These streams are then exchanged across several physical sites. Since all the streams capture pieces of the same scene, these streams show high spatial, temporal, and semantic correlations. Thus, DIMEs comprise of cyber-physical spatio-temporal correlated multi-streams.

In the past, multimedia traffic mainly consisted of single streams containing MPEG, avi, wmv videos which do not share any correlations with other streams. Thus, QoS definitions and resource management only depended on per stream basis. However, recent advancements in multimedia systems have witnessed a surge of highly interactive systems like Telepresence, Tele-immersion systems which use large number of correlated sensors producing highly correlated multi-streams in each session. Thus, correlated multi-streaming is an upcoming paradigm which needs to be tackled.
1.1 Problem Description

The presence of correlated multi-streams in the emerging multimedia systems open new avenues and complexities in 1) defining groups of correlated sets of streams, 2) defining and measuring Quality of Service across these correlated sets of streams, 3) correlated streaming of these correlated sets of streams, 4) correlated operations on these correlated sets of streams.

It is important to define a sound theoretical methodology to describe the correlations between the streams and group these streams into correlated sets. Also, currently, all the traditional QoS metrics measure the QoS on a per-stream basis, however, in a correlated multi-streaming scenario, these traditional metrics fail to capture QoS across multiple streams. Hence, new theoretical QoS model and new metrics definitions need to be provided to correctly capture QoS across multiple streams.

Based on the defined groups of correlated multi-streams, it is important to build a correlated streaming service which brings forth the properties to take into account when streaming correlated streams like the cross-dependencies between the streams, schedulability of the concurrent and dependent streams over the CPU cycles and the network, synchronization between these correlated multiple streams, and precedence based resource management between the correlated streams. Also, the presence of absolutely limited resources (both bandwidth, and computation) compared to the high resource demands of DIMEs stresses the need to focus on the correlations between the streams. Taking the correlations as important parameters, several correlated resource optimizations need to be designed and applied to these correlated multi-streams.

Thus, a complete end-to-end soft realtime correlated multi-streaming support needs to be designed and developed to realize these upcoming cutting-edge multimedia technologies. Tackling the above mentioned complexities plays an absolute key role in realization of high quality multi-streaming environments in DIMEs.

1.2 Contributions

Our contributions in this thesis are manifold:

1. Formalizing a concept of bundle of streams to define groups of correlated multi-streams
existing in state-of-the-art collaborative applications;

2. A multi-dimensional cyber-physical QoS model which is used to evaluate the bundle of streams;

3. Formalization of the Correlated Streaming as a Service (SAS) paradigm;

4. Design of a SAS Kernel based on the SAS paradigm with following properties:

   (a) Streams and bundles as first class objects,
   (b) Unified interface for diverse end-devices,
   (c) User-controlled runtime functions over streams and bundles,
   (d) Integrated session and bundle management,
   (e) Extensibility of SAS components

5. Quantitative validation of better performance of our QoS metrics against traditional QoS metrics;

6. Evaluation of bundle performance over three transport protocols;

7. Quantitative evaluation of SAS Kernel in a real DIME testbed.

1.3 Outline

The rest of the thesis is organized as follows: Chapter 2 gives an overview of the DIME systems and TEEVE (an example of DIME) system, and also covers the related work in the areas of this Bundle of Streams concept and Streaming as a Service Kernel. Chapter 3 presents the cyber-physical mapping between the physical sensory environment to hypothetical cyber sensory virtual world abstracting all the correlations and semantics between the sensory streams and consequently defines the concept of Bundle of Streams in this cyber-sensory world. In Chapter 4, a theoretical QoS model for bundle of streams is presented with two different sets of QoS metrics. Chapter 5 presents the correlated Streaming as a Service (SAS) paradigm, and the design of a real SAS-based SAS Kernel and Chapter 6 covers the implementation details of SAS Kernel in real DIME testbed and the system setup steps. Chapter 6 also covers the evaluation of both the major contributions; Bundle of Streams QoS Metrics, and Streaming as a Service Kernel and Chapter 7 concludes our research work with future directions in this space.
2.1 DIME Overview

Distributed Interactive Multimedia Systems (DIMEs) are advanced multi-sensory multimedia systems. DIMEs allow activity rich real-time collaboration across geographically distributed users. The remote users, and their personal spaces are immersed together in a common virtual world, where their collaborative activities appear as being performed in a common virtual world. With this, the distributed users experience belongingness to the common virtual world where they take part in exciting collaborative activities. DIMEs enable a plethora of interesting activities like interactive virtual gaming with real full body experience, fashion design, remote rehabilitation and physical therapy, cyberarcheology, sport activities, and dancing [1], [2], [3]. Figures 2.1(a), 2.1(b) show two exciting applications in DIMEs.

Figure 2.1: DIME Activities
DIMEs are composed of several networked components like sensing and actuating devices (e.g. cameras, microphones, body sensors, displays, haptic devices) and service gateways. DIMEs enable real-time interactive responses between these networked sensors producing high bandwidth and computation demanding streams. Thus, DIMEs provide architectural environments to support such highly interactive streaming applications across distributed components. Figure 2.2 shows the logical definition of DIMEs.

![DIME Definition](image)

The setup of each physical site in a typical DIME is shown in 2.3. It is composed of 3D-cameras arrays, microphone arrays, gateway servers, and 3D-displays. The 3D-cameras periodically capture current activities in DIME from different viewpoints, and these cameras individually produce realtime streams. These streams exhibit strong correlations both temporally and spatially. The temporal correlation is due to the fact that all cameras capture a logical snapshot of the information present at a time instant $t$ in a current DIME session, and temporal synchronization between these streams is required before showing the whole scene (obtained by stitching corresponding frames at same time instant $t$) on the display. The spatial correlation arises due to the relative positioning of the cameras in the physical space capturing the same scene from different viewpoints. Thus, streams from closeby cameras are clustered together to form discrete views $v_i$ covering 180 or 360 degrees of the physical space. Similar correlations occur between streams produced by other sensors like microphone arrays, body sensors, and haptic devices. Thus, highly correlated multi-streams are an inherent property of a DIME environment.
After capturing the activities, the produced streams are exchanged over Internet across remotely distributed participating sites. DIMEs make use of service gateways to exchange these streams in a soft real-time manner. Service gateways also support other important functionalities like overlay routing topology formation between the end sites, bandwidth management, QoS provisioning, multi-stream synchronization, and online monitoring. Figure 2.4 shows 3 DIME sites and the end-to-end connection setup between them. After receiving all the streams, renderers at each of the DIME site, use 3D vision algorithms to render remotely received streams together with local streams (from local DIME site) and this rendered image is then displayed on the 3-D displays.

## 2.2 DIME Systems: TEEVE

Several industrial systems can be categorized as DIMEs like Cisco Telepresence [4], HP Halo [5], and Technicolor gateways [6]. However, these systems have limited application domain as these focus on desktop type video conferencing applications. Emerging are the 3D Tele-immersion systems which provide a richer full body 3D experience with interactive activities across remote virtual environments.

TEEVE (Tele-immersive Environment for EVerybody) is a 3D Tele-immersion system devel-
Developed at University of Illinois, Urbana Champaign. TEEVE [7] is an example of DIME environment comprising of multiple remote sites collaborating in real-time. Figure 2.5 shows the TEEVE architecture.

2.2.1 TEEVE Architecture

Each of the remote sites comprises of 3D cameras, 3D displays, renderers and service gateways. 3D cameras perform the scene acquisition. The 3D cameras are placed at different viewing angles in a 180 degree plane and also at two different height levels (top and bottom). This allows for
a large visual coverage of the user’s body, enabling a more accurate 3D representation of the body. Currently, 4 3D Bumblebee2 (Point Grey Inc.) are setup in the TEEVE testbed. Due to the computational demands of 3D reconstructions, these 3D cameras are connected to camera rendering machines through IEEE1934 (Firewire) interface. These camera PC’s reconstruct the 3D video scene from the two/three eyes of the camera using vision based rendering algorithms. We use the vision software provided by University of California, Berkeley at the camera machines.

A hard-wired trigger is used to periodically capture synchronous frames across all the cameras. The trigger regularly emits a triggering signal with a frequency of 10Hz, resulting in a frame rate of 20fps per camera. The hardware trigger is controlled by a trigger software which runs on a separate PC. The trigger software receives a notification message everytime a camera grabs a 3D frame, and when notifications from all cameras are received, it sends the next trigger signal.

The streams produced from the camera PCs are delivered over LAN (in local site) to the service gateways. The gateway performs session and stream management between other sites. The session management tasks include session initiation, overlay routing topology formation between remote gateways, coordination, resource allocation, bandwidth management and congestion control. The stream management includes synchronization, encryption, and compression. The gateway acts as the aggregator and all the streams from remote sites are exchanged via the gateway servers only.
Figure 2.6 shows a 3 site configuration of TEEVE system.

Renderers are used to reconstruct complete 3D scene formed by rendering all the streams received from local/remote sites. The rendered frames are then displayed on the 3D/2D displays (42” plasma displays).

2.3 QoS Metrics for Synchronous Data Delivery

The related work can be classified in three broad areas: Metrics for Multimedia Systems, Consistency Models for Multimedia Systems, Quality of Service (QoS)

In the area of Metrics for Multimedia Application, Wijesekera et al. [8] [9], specify metrics to evaluate continuity and synchronization in Continuous Multimedia systems. To evaluate the metrics, user perception quantified by the Likert scale is used. Another important study is the work of Claypool et al. [10] in which a multidimensional metric to evaluate the perceptual quality of multimedia applications is proposed. Also related to this area are the per stream (flow) metrics for evaluating per stream QoS on congestion control mechanisms proposed by the Transport Modeling Research Group (TMRG) [11]. These metrics are heavily used in the evaluation of multimedia systems.

In the area of Consistency Models for Multimedia Systems we mention the work of Qin et al. [12] in which a loose consistency model for continuous objects in which the state of an object in different sites is separated by a few units in time is proposed. Also, the work from Choukair et al. [13] is important as it proposes a consistency model for Virtual Reality Systems (a specific type of DIME). The paper presents full consistency and soft synchronization mechanisms with minimal rendering degradation.

In the area of QoS, Grenhaigh et al. [14], proposes a QoS architecture for collaborative virtual environments. They define the QoS requirements of the application in terms of levels of awareness across different sites.

Also related to our work is the QoS architecture proposed by Campbell et al. [15] in which the authors propose a system wide QoS architecture spanning across all the layers of the system and specify QoS mapping between the layers. The approach uses a synchronization mechanism called
'orchestration' [16] between different streams to achieve synchrony and a QoS model that considers throughput, end-to-end delay (EED), jitter and packet and bit error rates. In [17], a framework for QoS mapping across different layers of a multimedia system is proposed. The framework spans over concurrent network, transport and application layer measurements of various QoS metrics.

To compare, our QoS model introduces a QoS framework for correlated set of streams called bundle of streams and we define multi-dimensional cyber-physical QoS metrics to evaluate distributed interactive systems.

2.4 Streaming Service Provision

Streaming service needs support from several areas including service gateways, network services, and modular operating system kernels. We discuss the related work in each of these research areas.

The current service gateways can be classified as a) residential gateways, b) sensor gateways, c) streaming gateways. In [18], [19], [20], [21] general architectures for home gateways, supporting small set of functionalities like protocol translation, media transcoding, and admission control is presented. For sensor gateways, [22] uses proxy servers to integrate small sensors into ad hoc networks and provides multiple device interfaces, and data processing however it is tailored to the needs of small sensors. In [23], [24], [25] real-time streaming gateways are proposed mainly allowing media format conversion, synchronization, and protocol translations. Most of these gateways focus on a small subset of the requirements of streaming services and hence fail to provide major goals like user-configurable session management, multi-correlated streaming, complex policies like routing, cross-layer QoS provisioning, advanced monitoring, synchronization, and minimal device interfaces. Our approach on the other hand is designed to comply with the current demands of the multimedia streaming services and tackles all the correlated streaming requirements.

In network services, OSGi [26] is a java-based generic service platform for service providers to deliver services to end users at home networks. These services are provided by interested third parties and can be dynamically loaded into the service platform. The problem with low-level OSGi is that it is very cumbersome to build a real-time multi-correlated streaming service and session management over OSGi [27]. In [28], an adaptive network service selection agent is used to select
wireless connectivity from different providers suiting the user demands. It uses user configuration based service provisioning, however its service is restricted to only wireless connectivities. [29] presents rulesets to capture class of service policies contained in device configurations allowing consistent application of policies across distributed routers. Compared to this, our approach is at the application level not the router level and we mainly focus on streaming as service.

Our streaming service draws concepts from operating systems like the motivation behind user-defined functions in our approach is similar to that of leaving policies to the end-users as in [30]. The interface of run-time loading of functions in our approach is similar to that explored in [31] for record and replay tools. Also, our approach follows the architectural considerations like division between data manipulation and transfer control, and integrated layer processing as pointed out in [32]. In [33] resource containers, providing resource management over processes and threads for network servers is presented. Compared to this, our approach is at a higher level of abstraction spanning across sites, sessions and streams.
CHAPTER 3

BUNDLE OF STREAMS

3.1 Background

DIMEs comprise of several sensing and actuating devices (e.g. cameras, microphones, body sensors, displays, haptic devices) as shown in Figure 3.1. At each site, sensing devices periodically capture current activities in DIME, forming a logical snapshot of the data produced by cameras, microphones, and body sensors. In general DIME model, each sensing device produces a different stream of data representing one dimension of the multimedia environment. These streams are then exchanged across several physical sites. Since all the streams capture pieces of the same scene, these streams show high spatial and temporal correlations. We call such cyber-physical spatio-temporal correlated streams a Bundle of Streams.

Figure 3.1: DIME Environment
3.2 Cyber-Physical Mapping of Sensors to Correlated Streams

Each physical space denoted as $PS$ in DIMEs consists of a) users who perform tasks or roles b) sensors to capture the sensory information present in the environment c) display devices to present the captured sensory information to the users. Commonly, the sensors and display devices in a physical space are of different types, different ranges, and are placed at different positions inside the physical space. Typical sensors include cameras, microphones, human motion trackers, and wearable health monitors. Video display, speakers, and haptic devices are among the most typical display devices used in DIMEs. Thus, a physical space $PS$ can be modeled as: Let the number of users, sensory devices, and display devices present in a physical space $PS$ be denoted by $|U|$, $|SN|$, and $|DI|$ respectively, then we define a physical space as:

$$PS = \{ \{U_i, SN_j, DI_k\} | \forall i \in [0, |U|), \forall j \in [0, |SN|), \forall k \in [0, |DI|) \}$$

The sensors often produce data streams with heterogeneous frame sizes and rates. We denote the data stream produced by a sensor $SN$ as $S$. The logical data unit of a stream is frame produced at time $t$ (physical or logical), denoted as $f^S_t$. A stream is a sequence of frames in increasing order of timestamp as $S = (f^S_0, f^S_1, \ldots, f^S_n)$. Thus, a DIME environment is composed of several data streams being exchanged between the collaborating sites. All the notations used in the paper are also given in Table 3.1.

To model the DIME environment, we map the physical spaces to a joint virtual cyberspace denoted by $CS$ which consists of all the streams being produced by all the sensors in the physical spaces collaborating in a session $SID$. Thus, each sensor belonging to a physical space gets mapped to one stream in the cyber space. A cyberspace also maps the properties of each stream like frame rate, frame size, resolution which depend on the sampling rate and range of the sensor in the physical space. The correlations between the physical sensors are also mapped to the streams in the cyber space. Figures 3.2 shows the mapping from physical spaces to cyberspace.

Types of correlations observed between streams are as follows: (1) spatial correlation: The physical positions of sensors may have impact on the spatial correlation between their streams.
<table>
<thead>
<tr>
<th>Task</th>
<th>Role</th>
<th>Type</th>
<th>Location</th>
<th>Range</th>
</tr>
</thead>
</table>

**Physical TI Spaces**

![Physical TI Spaces Diagram]

**Cyber TI Spaces**

- Streams
  - Properties:
    - Source
    - Frame rate
    - Size
  - Correlations:
    - Temporal
    - Spatial
    - Precedence

- Correlations:
  - Properties
  - Frames
  - Deadline

**Figure 3.2: Mapping from Physical Space to Cyber Space**

For example, the video stream that captures the upper body of a user is correlated with the video stream that captures the lower body. (2) **temporal correlation**: The physical phenomenon being monitored also causes temporal correlation between consecutive observation of sensors and between sensory information captured by different sensors (synchronization issues). (3) **precedence correlation**: Some streams can be assigned higher priority depending on the type/importance of the streams. For example, the video streams which contain the side view of the user are less important than the video streams which contain the front view of the user.

To model a cyberspace $CS$, we map each sensor $SN_j$ from each physical space $PS_i$ collaborating at a given time $t$ to a corresponding stream $S_{jPS_i}$ in $CS$. Once all the streams are added to the $CS$, a correlation function $cor()$ is used to derive the correlation between all the streams belonging to $CS$ as shown in 3.3.

Let the number of physical spaces collaborating in a given session $SID$ in $CS$ be denoted by $N^{CS}_{SID}$ and total number of sensors belonging to a physical space $PS_i$ be $|SN^{PS_i}|$, then $CS$ is
Figure 3.3: Cyber-Physical Correlations
defined as follows:

\[
CS = \{S_j^{PS} | \forall i \in [0, N_{SID}^{CS}) \text{ and } \forall j \in [0, |SN^{PS_i}|) \}
\]

\[
cor_{CS}(S_j, S_k) = \langle \text{spatial}_{cor}(S_j, S_k), \text{temporal}_{cor}(S_j, S_k), \text{precedence}_{cor}(S_j, S_k) \rangle;
\]

\[
\forall j, k \in [0, |CS|)
\]

3.3 Bundle of Streams Concept

To define the QoS metrics on cyber-physical streams, we first define a mechanism to find groups of streams which need to be considered together given the spatial and temporal correlations. We introduce the concept of **Bundle of Streams** wherein streams belonging to a common physical space and exhibiting high correlations belong to the same bundle. We present the definition of **Bundle of Streams** as follows:

**Bundle of Streams**: A Bundle of Streams is a collection of highly correlated cyber-sensory streams captured in the same physical space \(PS\) during the same DIME session \(SID\):

\[
B_{SID}^{PS} = \{S_j^{PS} | cor_{CS}(S_i^{PS}, S_j^{PS}) \geq threshold_{cor} \forall i, j \in [0, |SN^{PS}|) \}
\]

The definitions of function \(cor()\) and \(threshold_{cor}\) are application-specific and depend upon the system parameters and types of sensors deployed in the system.

**Example of Bundle of Streams**: Figure 3.4 shows three 3DTI sites, \(PS_1\), \(PS_2\), and \(PS_3\) collaborating with each other in session \(SID_t\). As shown, \(PS_1\) contains two cameras \(SN_1\), \(SN_2\), one audio device \(SN_3\), and one body sensor \(SN_4\). The bundle at site 1 is defined as \(B_{SID_t}^{PS_1} = \{S_1^{PS_1}, S_2^{PS_1}, S_3^{PS_1}, S_4^{PS_1}\}\). It may be noted that in cases of low correlations, not all sensory streams
There are two types of Bundles of Streams:

1. Strictly Correlated Bundle of Streams
2. Loosely Correlated Bundle of Streams

**Strictly Correlated Bundle of Streams** A Strictly Correlated Bundle of Streams is a collection of highly correlated sensory streams, wherein all the belonging sensory streams need to be fully synchronized with each other.

A good example of strictly correlated bundle of streams is the bundle formed on sensory streams produced at the same DIME site. These streams show very high spatial, temporal, and semantic correlation between themselves. In Figure 3.4, two strictly correlated bundle of streams \( B^{PS_2}_{SID_t} \), \( B^{PS_3}_{SID_t} \) are shown as only a subset of highly correlated sensors are present in the respective bundles.

**Loosely Correlated Bundle of Streams** A Loosely Correlated Bundle of Streams is a collection of loosely correlated sensory streams, wherein all the belonging sensory streams need to be loosely synchronized with each other.
A good example of *loosely correlated bundle of streams* in DIME environment is the hierarchical bundle defined on a receiver formed by grouping together strictly correlated bundles from the remote sites. Since the sites collaborate with each other hence, there is presence of loose spatial, temporal, and semantic correlations between these bundles.

In Figure 3.4, \( B_{S1D_i}^{PS_1} \) shows an example of loosely correlated bundle of streams as all sensors producing loosely correlated data are grouped together in a bundle.

### 3.4 Bundle of Bundles

Apart from intra-site correlations between sensory streams, there exist further correlations between inter-site streams. Typical types of correlations between inter-site streams include temporal (synchronization and consistency issues) and precedence correlations (importance of one site over the other). Thus, we define a hierarchical bundle on top of the bundles received from different physical spaces as follows:

**Bundle of Bundles**: A Bundle of Bundles is a collection of highly correlated bundles of streams belonging to different physical spaces \( PS_1, \ldots, PS_n \) during same DIME session \( SID \):

\[
B_{SID}^{PS_1, \ldots, PS_n} = \{ B_{SID}^{PS_i} | \text{cor}^CS(B_{SID}^{PS_i}, B_{SID}^{PS_j}) \geq \text{threshold}_{cor}^' \forall i, j \in [0, n) \}
\]

In Figure 3.4, \( B_{S1D_i}^{PS_2, PS_3} = \{ B_{SID_1}^{PS_2}, B_{SID_1}^{PS_3} \} \) shows an example of bundle of bundles defined across sites.

### 3.5 Operations on Bundles

In fully functional DIME sessions, several operations need to be executed in physical and cyber spaces along various stages of a) initialization b) running c) maintenance and d) shutdown. These operations are necessary for the functioning of the DIME sessions, and define tasks to be performed on objects (e.g. session, stream, sensor, user) composing the DIME environment. For example, common operations on a session include *AddUser, RemoveUser, QuerySession, Allo-*
cateResources, and MonitorResources. As we are adding a new object bundle to the DIME environment, we cover set of operations that can be applied to a bundle. Using these operations a bundle can be initialized, modified, queried, and deleted. Common operations on a bundle are as follows:

1. *AddStream*(\(B^{PS_i}, S\)) : \(B \times CS \rightarrow B\), where \(B^{PS_i} \in B\), \(S \in CS\), and \(AddStream(B^{PS_i}, S) \in B\); \(AddStream(B^{PS_i}, S) = B^{PS_i} \cup \{S\}\). This operation allows adding new streams to a bundle.

2. *DeleteStream*(\(B^{PS_i}, S\)) : \(B \times CS \rightarrow B\), where \(B^{PS_i} \in B\), \(S \in CS\), and \(DeleteStream(B^{PS_i}, S) \in B\); \(DeleteStream(B^{PS_i}, S) = B^{PS_i}/S\). This operation allows deleting streams from a bundle.

3. *EditStream*(\(B^{PS_i}, S\)) : \(B \times CS \rightarrow B\), where \(B^{PS_i} \in B\), \(S \in CS\), and \(EditStream(B^{PS_i}, S) \in B\); \(EditStream(B^{PS_i}, S) = edit(S, MD^S)\). This operation allows editing the stream and its meta-data \(MD^S\) containing properties and correlations like frame rate, size, resolution, spatial, temporal, and precedence correlations of a stream belonging to a bundle.

4. *QueryBundle*(\(B^{PS_i}, Q_j\)) : \(B \times Q \rightarrow CS\), where \(B^{PS_i} \in B\), \(Q_j \in Q\), and \(QueryBundle(B^{PS_i}, Q_j) \in CS\); \(QueryBundle(B^{PS_i}, Q_j) = \{S_i : i \in |B^{PS_i}|\}\). This operation allows querying a bundle with query \(Q_j\) and obtain set of streams which match the query.

### 3.6 Macroframes

After defining Bundle of Streams, we define a single cross-cutting entity on streams belonging to the same bundle on which the QoS metrics are applied. We define the concept of Macroframe as a logical snapshot of a bundle at time \(t\). We formally define Macroframe as under:

*Macroframe*: A Macroframe is a set of frames/information representing a time-slice of a bundle.
at time $t$ and/or within a small time interval $(t, t + \delta)$:

$$F_t^B = \{ f_{t_i}^S | \forall i \in [0, |B|); \forall t_j \in [t, t + \delta) \}$$

**Figure 3.5: Representation of a Macroframe in a DIME Architecture**

The number of frames composing a macroframe can differ in each macroframe depending on the relative rates of the streams and the time interval of the time-slice. For cases wherein a frame spans over more than one time-slice interval, it is considered to belong to only the macroframe defined on the earliest time-slice among them.

*Example of Macroframe:* In Figure 3.5, macroframe for time-slice $t_1$ is:

$$F_{t_1}^B = \{ f_{t_1}^{S_1}, f_{t_1}^{S_2}, \ldots, f_{t_1}^{S_N}, f_{t_1+\delta}^{S_N} \}.$$
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>cyberspace</td>
</tr>
<tr>
<td>PS</td>
<td>physical Space</td>
</tr>
<tr>
<td>Si</td>
<td>stream i</td>
</tr>
<tr>
<td>SipS</td>
<td>stream i ∈ physical space PS</td>
</tr>
<tr>
<td>fSi</td>
<td>frame ∈ stream S at time t</td>
</tr>
<tr>
<td>SN</td>
<td>a sensor</td>
</tr>
<tr>
<td>sid</td>
<td>session id</td>
</tr>
<tr>
<td>NsiD</td>
<td>total number of physical spaces in CS in session SID</td>
</tr>
<tr>
<td></td>
<td>total number of sensors in the physical space PSi</td>
</tr>
<tr>
<td></td>
<td>total number of streams in the cyberspace CS</td>
</tr>
<tr>
<td>corCS(Sj,Sk)</td>
<td>function to calculate amount of correlation between two streams ∈ CS</td>
</tr>
<tr>
<td>De(fSi)</td>
<td>delay on frame fSi</td>
</tr>
<tr>
<td>Ji(fSi,fSi+1)</td>
<td>jitter between two frames ∈ S</td>
</tr>
<tr>
<td>Sk(fSi,n,fSi+1,n,...,fSi,n)</td>
<td>skew between frames across different streams</td>
</tr>
<tr>
<td>LoI(S)</td>
<td>total loss of information on S</td>
</tr>
<tr>
<td>NumfLSN(S)</td>
<td>total number of frames ∈ S lost in network</td>
</tr>
<tr>
<td>NumfLE(S)</td>
<td>total number of frames ∈ S lost due to error</td>
</tr>
<tr>
<td>NumfLD(S)</td>
<td>total number of frames ∈ S lost due to expired deadline</td>
</tr>
<tr>
<td>NumfLrI(S)</td>
<td>total number of frames ∈ S lost due to incomplete macroframe</td>
</tr>
<tr>
<td>BsiD</td>
<td>bundle of streams originating in PS for DIME session SID</td>
</tr>
<tr>
<td>threshold_cor</td>
<td>threshold on the amount of correlation required between two streams to belong to the same bundle</td>
</tr>
<tr>
<td>BsiD,...,BsiD</td>
<td>bundle formed on bundle of streams originating in PSi</td>
</tr>
<tr>
<td>corCS(BsiD,BsiD)</td>
<td>function to calculate amount of correlation between two bundles</td>
</tr>
<tr>
<td>threshold_cor′</td>
<td>threshold on the amount of correlation required between two bundles to belong to a bundle of bundles</td>
</tr>
<tr>
<td>FtB</td>
<td>macroframe occurring at time t in bundle B</td>
</tr>
<tr>
<td>BS</td>
<td>total number of streams ∈ bundle B</td>
</tr>
<tr>
<td>FtB</td>
<td>total number of frames ∈ macroframe FtB</td>
</tr>
<tr>
<td>DeB(FtB)</td>
<td>bundle delay on macroframe FtB</td>
</tr>
<tr>
<td>JiB(FtB,FtB+1)</td>
<td>bundle jitter between two consecutive macroframes</td>
</tr>
<tr>
<td>SkB(FtB,0,...,FtB,n)</td>
<td>bundle skew between macroframes across different bundles</td>
</tr>
<tr>
<td>LoIB(B)</td>
<td>bundle total loss of information on B</td>
</tr>
<tr>
<td>ThB(B,tp)</td>
<td>bundle throughput for bundle B with tp as the unit time</td>
</tr>
<tr>
<td>tp</td>
<td>time period for bundle throughput calculation</td>
</tr>
<tr>
<td>Np</td>
<td>number of streams dealt as priority streams</td>
</tr>
<tr>
<td>P(B,Np)</td>
<td>set of priority streams from bundle B</td>
</tr>
<tr>
<td>WDeB(FtB)</td>
<td>weighted macroframe delay</td>
</tr>
<tr>
<td>WJiB(FtB,FtB+1)</td>
<td>weighted macroframe jitter</td>
</tr>
<tr>
<td>WSkB(FtB,0,...,FtB,n)</td>
<td>weighted skew between macroframes across sites</td>
</tr>
</tbody>
</table>

Table 3.1: Table of Notations for Bundle of Streams.
CHAPTER 4

BUNDLE OF STREAMS QOS METRICS

4.1 Background

The QoS metrics defined in the literature like delay, jitter, loss rate are defined on a per stream basis and hence, cover only the relationships within a frame and between frames of a stream. These traditional QoS metrics are defined as follows:

\[ De(f^S_t) = t_{\text{recv}}(f^S_t) - t_{\text{send}}(f^S_t) \]
\[ Ji(f^S_{ti}, f^S_{ti+1}) = |De(f^S_{ti+1}) - De(f^S_{ti})| \]
\[ Sk(f^S_t, f^S_i, \ldots, f^S_n) = \max_{i,j}|De(f^S_i) - De(f^S_j)|; \quad \text{where } 0 \leq i, j \leq n, i \neq j. \]
\[ LoI(S) = \text{Num}_{f_{\text{LN}}}(S) + \text{Num}_{f_{\text{LE}}}(S) + \text{Num}_{f_{\text{LD}}}(S) \]

Please refer to Table 3.1 in Chapter 3 for notations. These metrics lack the ability to be applied to currently emerging cyber-physical streams which bear high spatial and temporal correlation with other streams as well as temporal correlation within themselves. Thus, for new collaborative applications with several correlated integrated streams, we define a new QoS model defined over Bundle of Streams.
4.2 Cyber-Physical Non-Weighted QoS Metrics

Given the lack of QoS metrics for measuring the quality and performance in multi-dimensional correlated streams, we extend the current QoS model by defining cyber-physical spatio-temporal QoS metrics to handle integrated streaming. The cyber-physical spatio-temporal QoS metrics are unique in the sense that they are computed over two dimensions (both space and time) where spatial position of streams in cyber-physical space add spatial dimension and the temporal relationship between streams in cyber-physical space add the temporal dimension.

4.2.1 Bundle Delay

The DIME environments involve high level of user interactivity between different sites and hence, are very sensitive to overall delay experienced. In a bundle of streams, total delay not only comprises of the delay experienced on each stream in the bundle but also the time taken to complete a macroframe. Thus, delay for a bundle is a two-dimensional metric, with both temporal and spatial aspects in it.

**Bundle Delay:** For a given macroframe, the Bundle Delay $De^B$ is the difference between the time at which the first frame belonging to the macroframe is sent at the sender and the time at which the last frame is received at the receiver as shown below:

$$De^B(F^B_t) = \max_i t_{recv}(f_i) - \min_i t_{send}(f_i);$$

where $0 \leq i < |F^B_t|$.

Figure 4.1 shows an example of the bundle delay between two sites on bundle $A$ with $De^A(F_i) = t'(f_n) - t(f_1)$.

4.2.2 Bundle Jitter

The DIME environments are composed of several sensory streams which are highly sensitive to jitter like audio and video streams. High jitter between macroframes can lead to poor overall quality. The formal definition of Bundle Jitter is as follows:
**Bundle Jitter:** For a given bundle, the Bundle Jitter $J_i^B$ is the variation in BundleDelay between two consecutive macroframes:

$$J_i^B(F_{ti}^B, F_{ti+1}^B) = |De^B(F_{ti+1}^B) - De^B(F_{ti}^B)|$$

From Figure 4.1, for bundle $A$, the $J_i^A(F_{ti}^A, F_{ti+1}^A) = |De^A(F_{ti+1}^A) - De^A(F_{ti}^A)|$.

### 4.2.3 Bundle Skew

DIMEs consist of highly interactive real-time sessions among various users across geographically distributed areas, for example, playing virtual games to dance performances. In this, at any given instant $t$, a player/dancer needs to see a consistent view of all the remote sites, requiring synchronization of all the frames produced at time $t \pm \delta$ at all the sites. Thus, differences in delays across sites can reduce the responsiveness of the system. For such scenarios, the skew between macroframes across several sites is of importance.

**Bundle Skew:** Bundle Skew is defined as the maximum difference in BundleDelay of macroframes.
belonging to different bundles of streams received from different sites:

\[
S_k^B(F_t^{B_0}, F_t^{B_1}, \ldots, F_t^{B_n}) = \max_{i,j} |D_e^B(F_t^{B_i}) - D_e^B(F_t^{B_j})|;
\]

where \(0 \leq i, j \leq n, i \neq j\).

From Figure 4.1, the \(S_k^B = \max(|D_e^A(F_i^A) - D_e^B(F_i^B)|, |D_e^A(F_i^A) - D_e^C(F_i^C)|, |D_e^B(F_i^B) - D_e^C(F_i^C)|)\)

### 4.2.4 Bundle Loss of Information

Due to the large amount of data to be streamed across several sites, the DIME environments often face scarcity of network bandwidth. To adapt to the dynamics of the bandwidth fluctuations, DIMEs apply congestion control mechanisms. Common Application layer congestion control mechanisms for DIMEs include reducing frame rate by dropping frames, dropping non-priority streams, etc. Part of stream/frames can also get corrupted or lost while in transit. Both the above cases lead to loss of information from the receiver’s perspective and hence, reduces the quality of the output at the receiver. Thus, we define bundle loss of information as follows:

**Bundle Loss of Information:** The total Bundle Loss of Information comprises of (a) intermittent lost, dropped, or delayed frames belonging to a stream (b) all the streams which are lost, dropped, or delayed (in this, we consider all frames belonging to these streams as lost.) (c) frames which depend on the lost, dropped, or delayed frames (belonging to same macroframe) in (a) and (b) and need to be dropped due to incompleteness. The unit of measurement is number of frames and bundle loss of information is defined as follows:

\[
LoI^B(B) = \sum_{0 \leq i < |B|} Numf_{LN}(S_i) + Numf_{LE}(S_i) + Numf_{LD}(S_i) + Numf_{LI}(S_i)
\]
4.2.5 Bundle Throughput

DIMEs are composed of highly dynamic heterogeneous streams with different frame rates across streams, variable number of bits per frame, and variable frame rate per stream. The granularity of the data is not at bit level but is at macroframe level i.e. end users see either the completed macroframe or none of it. Thus, the bundle throughput is defined as:

**Bundle Throughput:** For a given bundle, the total size over the completed macroframes received per unit time \( t_p \) is the Bundle Throughput:

\[
T_{th}^B(B, t_p) = \sum \frac{\text{size}(F^B_i)}{t_p}; \text{ where } t_{current} - t_p \leq t_{recv}(F^B_i) < t_{current}
\]

4.3 Cyber-Physical Weighted QoS Metrics

**Observation 1:** Based on the criticality of content, different streams can have different importance. For example, in [34], [35], the relative priorities are defined based on the contribution factor of streams towards the overall scene. The presence of prioritized streams directly influences how the QoS metrics need to be calculated. Any QoS disturbance on camera streams capturing the front view of the 3D model have greater impact on the overall quality of the rendered scene against QoS disturbance on camera streams capturing the side views.

**Observation 2:** Higher priority streams are allocated more resources to minimize delay, jitter, etc as compared to lower priority streams. Thus, priority streams may be rendered with faster rate. This leads to loose coupling between the frames.

Thus, depending on the relative importance of streams we define *weighted cyber-physical QoS metrics*. It is important to mention here that the inclusion of relative priorities of the streams add additional dimension to the spatio-temporal metrics defined in section 4.2, and hence, weighted cyber-physical metrics are truly a multi-dimensional metrics.
4.3.1 Weighted Bundle Delay

Based on the relative priorities given by $\text{precedence\_cor}(S_j, S_k)$, we bifurcate the bundle of streams into two sets a) High priority streams, and b) Low priority streams. For a given bundle, we pick first $N_p$ number of streams from a list of sorted priority streams, where $N_p$ is derived based on the characteristics of specific DIME and the dynamics of the system like current availability of resources. These $N_p$ streams form a bundle of streams on which the bundle delay is computed as described below:

**Weighted Bundle Delay:** For a given bundle, let $P(B, N_p) = \{S_i^B \mid \forall i \leq N_p; p(S_i^B) \geq p(S_{i+1}^B)\}$ represent a bundle of priority streams; thus, the weighted bundle delay is the bundle delay experienced over the priority bundle $P$:

$$WDe^B(F^P_t) = \max_i t_{\text{recv}}(f_i) - \min_i t_{\text{send}}(f_i);$$

where $0 \leq i < |F^P_t|$.

4.3.2 Weighted Bundle Jitter

The weighted bundle jitter signifies the variation in the weighted bundle delay experienced in completing consecutive prioritized macro-frames:

**Weighted Bundle Jitter:** For a given bundle, set of priority streams $P$, the weighted bundle jitter is the variation in weighted bundle delay between two consecutive macroframes belonging to $P$:

$$WJi^B(F^P_{ti}, F^P_{ti+1}) = |WDe^B(F^P_{ti+1}) - WDe^B(F^P_{ti})|$$

4.3.3 Weighted Bundle Skew

Similar to defining priorities for individual streams, priorities can be defined for individual sites based on the importance of the data being produced at the individual sites. These priorities can be defined on a system wide scale or on each receiver site.

**Weighted Bundle Skew:** Weighted Bundle Skew is defined as the maximum difference in Weighted
Bundle Delay of macroframes received from different sites:

\[ WS_k^B(F_t^{B_0}, F_t^{B_1}, \ldots, F_t^{B_n}) = \max_{i,j} |WDe^B(F_t^{B_i}) - WDe^B(F_t^{B_j})|; \]

where \(0 \leq i, j \leq n, i \neq j.\)
CHAPTER 5

SAS KERNEL: BUNDLE STREAMING AS A SERVICE KERNEL FOR CORRELATED MULTI-STREAMING

5.1 Background

In DIMEs, cameras, microphones, body sensors, and haptic devices act as the input devices and displays, speakers, and actuators act as the output devices. DIMEs make use of service gateways to transfer content from input devices to remote output devices over the Internet. Apart from streaming, DIME service gateways need to provide various functionalities like overlay routing, bandwidth management, QoS provisioning, synchronization, and monitoring.

Several architectures for streaming gateways interconnecting local area networks like Ethernet, Bluetooth, wireless access to the Internet exist in the literature. However, these service gateways [21], [22], [20], [23], [36] are limited to functionalities like relaying, multiplexing, translating, and managing local resources only, which fail to satisfy the requirements of DIMEs. There are some proprietary gateways developed at HP Halo [5], Cisco Telepresence [4], and Technicolor [6], however they are tailored to cater closed applications and their internal functionalities are publicly unknown.

DIME requirements differ from those of traditional gateways as:

1. Presence of multiple correlated sensors interacting in a DIME session requires support for large scale correlated multi-streaming as an inherent functionality.
2. High interactivity in DIME sessions requires soft real-time delivery of all streams.
3. Advanced QoS services across multiple spatially and temporally correlated streams need to be supported.
4. With end-devices dispersed across remote locations, distributed resource management is important in DIMEs.
5. Lack of standard stream formats across sensors from diverse vendors require \emph{unified interface} with the end-devices.

Based on the above requirements, concepts of \emph{correlated multi-streaming}, \emph{device abstraction}, and \emph{user-controlled services} need to be supported in DIMEs. Other important challenges include \emph{flexibility}, and \emph{scalability} of DIME frameworks.

In this thesis, we envision a novel paradigm, \emph{Bundle Streaming as a Service (SAS)} to model correlated multi-streaming service, where correlated multi-streams, also called \emph{bundle of streams} are first class objects. We propose a SAS-based, generalized, distributed service kernel, \emph{SAS Kernel} to setup, process, and control bundles of streams. We emphasize that SAS and SAS Kernel are not limited to just DIMEs but provide a fundamental foundation of modern service-oriented architectures for wide range of stream-based applications (e.g., 3D Streaming).

\subsection*{5.2 Bundle Streaming as a Service (SAS) Paradigm}

We propose a generalized Bundle Streaming as a Service paradigm for platforms providing correlated soft real-time multi-streaming as the key service. The guiding properties of SAS are as follows:

\begin{itemize}
\item \emph{Distributed Correlated Multi-Stream Support} - DIMEs are composed of distributed correlated multi-streams called \emph{Bundle of Streams (BoS)} [37] sharing high spatial and temporal correlations. These bundles interact in synchronous and soft real-time manner. The current streaming protocols like RTP/RTCP, SIP, RTSP do not take into account efficiently the spatio-temporal dependencies among large sets of streams. To deal with this, the SAS model inherently supports: (1) large scale correlated soft real-time streaming, (2) end-to-end session management based on media correlations [38].

\item \emph{Universal Open Access} - Unlike the Internet Protocols which have become the lingua franca, there is a lack of well-agreed formats across emerging devices like 3D cameras and microphone arrays. To overcome the problem of implementing large sets of formats, SAS model supports universal access policy with well-defined interfaces to the end-devices. In SAS,
varied types of streaming devices with different standards seemlessly connect and stream the data.

- **User-defined Functions** - SAS provides the flexibility of provisioning two types of run-time functions on streams: (1) system-defined functions like rate control, congestion control, and multi-stream synchronization, (2) user-defined functions like compression, encryption, and view management. These functions can be requested in an on-demand basis.

- **Availability** - It is anticipated that in the future, access to SAS will follow "always on" paradigm, like cable modem access is today. Thus, SAS is highly available at all times.

- **Robustness** - For SAS, the capability to monitor performance, isolate faults, and automatically recover from faults is critical. The robustness of SAS comes through (1) On-demand monitoring services with varied resolutions, (2) Fault localization and easy recovery mechanism.

- **Scalability** - It is anticipated that larger scale of sensors will enhance the Quality of Experience (QoE) of users. Thus, SAS provides scalability in terms of supporting large sets of streams. Also, extensibility of a SAS-based framework is important to support adding new functional services as need arises.

Thus, the goal of the SAS is to foster bundles of streams needing correlated multi-streaming support, universal access across diverse devices, and user-controlled runtime functions in future DIME systems. To realize the SAS paradigm, we present SAS Kernel, set of real-time integrated services that enforce SAS properties (as outlined above) at runtime.

### 5.3 SAS Kernel Framework

In DIMEs, distributed end-devices share streams and resources in real-time collaborative sessions. The SAS Kernel provides run-time system for easy setup, processing, management, and access of bundles of streams. The SAS Kernel implements the SAS properties as follows:
**Strong distributed correlated multi-streaming support:** SAS Kernel ensures correlated multi-streaming by (1) Managing and keeping states of streams, bundles, sessions, and resources, (2) Providing streaming policies for correlated soft real-time scheduling, co-operative congestion control, and overlay routing. The Management Entities (Figure 5.1) handle correlated multi-streaming.

**Universal access and easy availability:** SAS Kernel provides (1) Unified interface for end-devices, (2) Easily configurable stream specifications to describe device and stream characteristics. The SAS Interface (Section 5.5.1) provides universal access.

**Runtime mechanisms and functions:** (1) SAS Kernel follows the principle of *Separation of mechanism and policy* [39], i.e., the mechanisms only provide a unified framework for plugging-in the policies/functions and the actual functions are implemented at the user space, (2) SAS Kernel provides runtime loading of *user-defined functions* operating over streams or bundles. The Runtime Entities (Figure 5.1) and Function Manager (Section 5.5.2) provide functions and mechanisms. This design also allows easy availability.

**Robustness:** To ensure robustness (1) A cross-layer online monitoring interface is provided, (2) Runtime analysis of monitoring data to trigger recovery procedures is done. The Monitoring Manager (Figure 5.1) ensures robustness.

**Scalability:** SAS Kernel provides both device scalability and stream service extensibility via (1) User-configurable interfaces with varied end-devices, (2) Modular design of stream services to allow user-defined functions on streams, (3) Modular design of all SAS Kernel entities built on the principle of separation of concerns. The SAS Interface and Function Manager (Section 5.5) discuss the extensibility of the framework.

In SAS Kernel, the SAS properties get implemented as management, runtime, and monitoring entities in the session subsystem on top of a transport subsystem. Since streams are first class objects in SAS Kernel, each of the entities keeps track of and controls streams and stream derivatives (e.g., bundles, frames). Figure 5.1 shows the layout of various entities over the transport subsystem. A brief description of each of them is as follows:

**Management Entities:** The management entities manage sessions, bundles, streams, frames, and their corresponding resources. They provide mechanisms for generic tasks like overlay routing and provide interfaces to dynamically load the runtime entities. There are four management entities:
1. Session Manager - It performs session initiation, membership control, and session management. It takes management decisions and provides mechanisms to load session level functions like overlay routing and admission control.

2. Bundle Manager - It handles the correlation between the streams and defines the policies to group multiple streams into correlated bundles of streams. It provides mechanisms for runtime functions over these correlated bundles of streams like cooperative congestion control, prioritization, view management, and bundle of streams (BoS) metrics [37].

3. Stream Manager - It keeps states about receipt and delivery of streams across sites and determines policies for streaming. It categorizes streams as InStreams (from input devices) and OutStreams (to output devices). Mechanisms for stream-based runtime functions like compression, encryption are also provided by this manager.

4. Resource Manager - It manages overlay network resources like bandwidth and delay to ensure real-time delivery of streams.

**Runtime Entities:** The runtime entities provide specific system/user-defined policies for the
mechanisms like Mesh protocol for overlay routing. These entities are dynamically pluggable real-time functions operating over sessions, bundles, streams, frames, and network resources. These entities are open to be either implemented by SAS Kernel system-admins or the end-users. Examples of runtime entities at each level are shown in Figure 5.1.

**Monitoring Entity:** SAS Kernel implements a cross-layer event-driven monitoring entity. This entity provides real-time monitoring plane for overall system monitoring. The monitoring entity forms a feedback loop by communicating the states from the run-time functions to the corresponding managers, allowing the managers to take appropriate actions like adaptation, or policy switching. The monitoring entity also monitors for faults and failures.

**Transport Subsystem:** To ensure soft real-time delivery, the transport subsystem abstracts the underlying transport layer protocols allowing end-users to dynamically request appropriate protocols like TCP, UDP, DCCP based on application type and network conditions. The frames are encapsulated using our DIME specific S-RTP protocol (Section 5.5.1) which adds semantic information (used by managers) like stream type, functions requested, device addressing, and streams in same bundle.

### 5.4 Stream Flow in SAS Kernel

Distributed SAS Kernel is realized through a set of multiple distributed SAS gateways and SAS interfaces as shown in Figure 5.2.

SAS gateways take on the responsibility of hosting the SAS Kernel instances and the SAS interfaces (SASI) provide the connectivity between the end-devices and the SAS Kernel. We assume that all gateways and end-devices can be connected to each other via the Internet. Figure 5.3 shows the end-devices, SAS interfaces, and the functional placement of the SAS Kernel entities in a gateway. The streaming algorithm is as follows:

1. Session Initiation: A streaming end-device first starts a connection with the SAS interface present at the end-device machine. The SAS interface initiates a session with the closest SAS gateway and requests the services specified in the user-defined XML configuration. The request is handled by the Session Manager in the gateway. It verifies if the requested
services are supported and sends an *ACK* to the SAS interface. On positive *ACK*, Session Manager opens data and control connections with the end-device through the SAS interface. It also constructs overlay routing topology with other gateways, stores the meta-data about the new session, instantiates a Stream Manager for the joined stream, groups streams into bundles, and instantiates Bundle Manager.

2. **End-to-End Streaming:** An input device communicates its stream to the SAS interface. The SAS interface applies the S-RTP headers on each packet based on the information specified in the XML file. The packets are then sent over a chosen transport layer protocol to the corresponding InStream instantiated by the Stream Manager for this session. Once the InStream starts to get delivered in SAS Gateway, the Stream Manager creates corresponding sets of OutStreams based on number of requesting output devices. The InStreams are then connected to the respective OutStreams.

3. **Runtime Functions:** The run-time functions are loaded by the Function Manager (FM) present in each of the Managers. The InStreams and OutStreams are processed through the Bundle Manager to apply user-demanded bundle functions over bundles. The Stream Manager then applies stream based functions. For resource optimization, Resource Man-
4. Monitoring: Each entity implements hooks and callbacks to send monitoring information like QoS performance, resource utilization, and faults to the Monitoring Manager. Based on the received information, Monitoring Manager takes appropriate QoS or fault tolerance measures.

5.5 SAS Kernel Design

The two main components of SAS Kernel are SAS Interface and Kernel Function Managers which are discussed in detail in the following sections.
5.5.1 SAS Interface

The device-SAS Kernel interface provides universal open access (Section 5.3) and faces the challenges of (1) Multiple non-standardized stream formats of end-devices, (2) Requirement to understand all the stream formats to allow functions over streams.

The above challenges severely affect the scalability and flexibility of the service gateways. To address this issue, current solutions only implement a subset of these stream formats and thus, fail to support devices from diverse vendors. Instead, our approach relies on separating the stream formats from the main SAS Kernel using configuration mechanisms to specify the formats at runtime. Thus, SAS Kernel realizes four concepts: 1) End-to-End Tunneling, 2) Device Stream Specification, 3) Semantic data propagation through S-RTP, and 4) Service Negotiation.

End-to-End Tunneling

The idea behind SAS Kernel is that end-devices should interact agnostic of the SAS Kernel i.e. the end-devices do not know if they are communicating via SAS Kernel. The challenge in providing agnostic connection is that there should be no source code modification at the end-devices. To achieve this, POSIX socket API is used as an interface between end-devices and SAS Kernel.

Figure 5.4: Socket Interface and Tunneling

The assumption behind using socket API is that the end-devices in DIMEs mostly follow client-server type of connections and they usually provide interface to specify the IP and port number of
the remote device. Thus, the end-devices can be dynamically configured to connect to the SAS interface. The SAS interface, placed at each device, uses socket API to intercept the traffic from the input devices and send it via the SAS Kernel to the output devices. In addition, a peer-to-peer virtual tunnel is created between the devices where the virtual tunnel is supported by the underlying SAS Kernel. Figure 5.4 shows the socket interface and the tunnel.

From a networking point of view, SAS Interface and SAS Kernel form a Virtual Network between input and output devices. The SAS Interface listens and forwards all the communication from the input devices to the output devices via SAS Kernel. Data tunneling can be used at different layers of the TCP/IP stack. Our approach uses application level tunneling since it allows to keep most of the semantic information of the application layer session protocols.

**Device Stream Specification**

In order to apply functions on streams, SAS Kernel needs to understand the semantics of the stream, i.e. the packet structure. Thus, SAS interface requires end-users to provide a simple high-level specification of the stream semantics in a user readable language like XML. The specification is composed of two main parts: (1) Device Specification containing general metadata about the device, (2) Stream Specification containing stream format.

The Device Specification consists of a unique identifier for addressing the device in the originating site, the content type (e.g., video, audio), content subtype (e.g., for video point cloud, mesh) and the transport protocol the device uses (e.g., TCP, UDP). The Stream Specification specifies the format of the sequence of data packets as they appear within the stream. There are two general formats: fixed-size packets and variable-size packets. The fixed-size packets require only packet size to be specified while the variable-size packets require a fixed size header containing the packet size to be specified. Other stream parameters like frame rate, color information are specified through *Handshake packets* between the end-devices. This specification allows for marking packets as *Handshake packets*. SAS Kernel forms a multicast network between the input devices and the output devices, requiring storing and replaying these *Handshake packets* when new output devices are added to the kernel. The packet count specifies how many of each type of packets are present consecutively in the stream.
Figure 5.5: Device Stream Specification for a Video Stream

Figure 5.5 shows an example XML configuration file used in the 3D Tele-immersion system in our lab for a video stream. The camera protocol is comprised of single fixed handshake packet of 140 bytes followed by all (packet count of -1 indicates possibly infinite) payload packets of variable size that have a header of 10 bytes, with packet size specified at byte 6 in the header. Moreover, this specification is easy to implement and flexible enough to allow a wide range of end-devices to interface with our SAS interface without modification or recompilation.

**SAS Real-Time Protocol (S-RTP)**

Each data packet read by the SAS interface is then encapsulated using the SAS Real-Time Protocol. S-RTP is similar to RTP but it is tailored to include DIME specific session semantics and lighter-weight. Through S-RTP, session semantics like device addressing, services requested, and groups of streams forming bundles are marked on each packet, allowing easy dissemination of each stream’s state to all SAS components.

The structure of the S-RTP packet is shown in Figure 5.6. The packet first specifies the version
The stream type and subtype together form a tuple to uniquely identify the type (video, audio, sensory data) and the data format (e.g., for video, mesh and point-cloud). Next, S-RTP packet contains a list of all stream IDs forming a bundle `<BundleList>`, timestamp of packet creation, fixed/variable payload, and frame number.

### Service Initiation and Negotiation

After reading the stream specification and constructing an S-RTP packet, the SAS interface at the joining end-device initiates a session with the SAS Kernel. The Session Manager in the SAS Kernel handles the session initiation and service negotiation tasks. Remote procedure calls (RPC) and marshalling is used between SAS interface and SAS Kernel and simple session initiation and negotiation protocols are used as can be found in the literature. Our contribution in SAS Kernel is that the SAS Kernel allows dynamic pluggability of different correlations based admission control and bundle routing algorithms as need arises in the session and resource management.

The SAS interface sends a `JOIN` request message specifying desired transport protocol to use, the characteristics of the bundles and joining streams (e.g., periodic or aperiodic, variable or fixed packet sizes, payload type, payload sub-type, expected bandwidth usage), and the services requested (encryption, compression, congestion control). Upon receipt of the `JOIN` message, the
SAS Kernel verifies whether it can support services requested, and if so, opens required data ports and returns an ACK containing the ports. The SAS Kernel renegotiates if it does not support any of the services with the SAS interface. The Session Manager in SAS Kernel then creates InStreams and Bundles accordingly, bookmarks the parameters, and uses the data channels for data transfer.

In case of output end-device join, the payload type and subtype tuple provides a hierarchical way for the SAS Kernel to determine which bundles should be routed to the output device by matching the payload type and sub-type of the possessed InStreams with those specified. For example, one may use two renderers to display the frontal and back camera streams respectively; although they all identify the “video” type, one renderer and the frontal cameras use the “frontal” sub-type, and the other renderer and the back cameras use the “back” sub-type.

The strategy for interconnection and exchange of streams between the SAS gateways depends on the chosen routing protocol in the SAS Kernel. Some useful routing protocols for DIMEs are application level multicasting [40] like Viewcasting, Mesh. The discussion of these routing algorithms is out of scope of this paper.

5.5.2 SAS Kernel Function Manager

In order to provide runtime stream-processing functions, i.e., user controllable functions (as discussed in Section 5.3), each manager in SAS Kernel implements a Function Manager (FM) as shown in Figure 5.3. The function manager is responsible for: (1) Implementing mechanisms, and (2) Scheduling functions on bundles, streams, and frames.

To support extensible operations, SAS Kernel divides the execution plane in two spaces: End-User Space and System Space. User-controllable functions are in the End-User Space while all the other SAS Kernel functions and resource management remain in the System Space. New functions to be added to the SAS Kernel are compiled separately by end-users into dynamically linked libraries and these functions are loaded and linked at runtime by the FM. Functions interact with FM using system calls (Syscalls) and FM uses Upcalls to the functions. Figure 5.7 shows FM architecture.

The Syscalls provide direct access to the bundle and stream meta-data, S-RTP packet format, and also to the raw payload implemented by the end-devices. Each function implements an object and
FM keeps the state information, allocates memory and forks threads. This makes FM suitable for supporting parallel concurrent functions. Functions are executed as a computing pipeline where the user can configure the order in which the operations are applied. A scheduler inside FM is responsible of context switching to the corresponding operation. FM thus provides the support for defining mechanisms at each level of data abstraction and load user-specific functions to implement these mechanisms. This ensures high extensibility of services in SAS Kernel.
CHAPTER 6

IMPLEMENTATION AND EVALUATION RESULTS

6.1 Evaluation of Bundle of Streams Concept in DIMEs

To evaluate our Bundle of Streams QoS model we use the TEEVE (TeleImmersion for Everybody) system, a real DIME system [7]. We first justify the need for new QoS metrics by evaluating the performance of traditional metrics in real DIME system. As a case study we use the metrics in our QoS model to compare performance of this TI system at the bundle concept under three transport protocols: Transmission Control Protocol (TCP), Datagram Congestion Control Protocol (DCCP)[41] and User Datagram Protocol (UDP).

6.1.1 Data Model

The TEEVE system is composed of multiple Tele-immersive (TI) sites. Each site is composed of cameras, displays and service gateways. Each stereo camera produces a 3D video stream with variable frame size in the range of 5KB to 30 KB, and variable frame rate ranging between 10 to 20 frames per sec. To make the experiments repeatable, we use a recorded creative dance performance with large amount of movements within the captured videos. Four cameras simultaneously record the dance performance which is later regenerated and streamed within the participating sites. Bi-directional data communication is used between all the participating sites.

6.1.2 Network Model

For each experiment we use 3 TI sites each at, University of Illinois at Urbana-Champaign (UIUC), Centrum Wiskunde & Informatica (CWI) in Amsterdam, and University of California at Berkeley.
Figure 6.1 shows the experimental setup with these three sites. Internet 2 based communication channel is used between UIUC and UCB and normal every-day internet connection is used between UIUC to CWI for streaming. The data plane comprises of service gateways exchanging streams directly with each other and the end-renderers which render the macroframes together and show it on the end-displays. The control plane includes service gateways and a central session controller which performs overlay streaming topology formation between the service gateways. No extra assumptions are made on the internet behavior in any of these studies.

![Experimental Setup with 3 Remote Sites for Bundle of Streams Evaluation](image)

6.1.3 Software and Machine Configuration

We use BumbleBee2 stereo cameras (Point Grey Inc.) as 3D cameras, and camera software from University of California, Berkeley with hard-wired trigger is used to capture synchronous frames across 4 cameras used. Each of the three sites spawn a service gateway to exchange real-time TI streams. The gateway-2.0 version of the gateway software developed at UIUC is used at each of the service gateways. This version of the gateway software allows capabilities for TCP, DCCP, and UDP connections with the other gateways and supports mesh based topology formation between the remote sites using a central session controller. The session controller controls the topology formation through the control channel and is run on the UIUC gateway server. The renderer
software from University of California, Davis is used to show the final rendered macroframes on the displays.

For the gateway server, we use 4 Dell Precision 670 with dual Intel Xeon processor at UIUC, DELL Optiplex 780 at UCB and AMD Athlon 64 Processor 3500+ at CWI.

6.1.4 Evaluation of Traditional QoS Metrics vs Cyber-physical QoS Metrics

To justify the need of new QoS metrics, we evaluate the performance of traditional QoS metrics in DIME environment. A bundle consisting of two streams each is sent from CWI and UCB to UIUC. Figure 6.2(a) shows the delay experienced on each stream vs the delay experienced on a bundle. The delay on each stream only reflects a small portion(typically 55% to 65%) of the total delay experienced on a bundle. Thus, this signifies need of new metric to accurately quantify the delay DIMEs. Similarly, Figure 6.2(b) shows the inaccuracy of jitter calculated on each stream as compared to the actual jitter experienced on macroframes.

Figure 6.2: Comparison of Cyber-Physical Metrics against Traditional Metrics
6.1.5 Case Study: Evaluation of TCP, DCCP, UDP on Bundle Metrics

We apply our cyber-physical QoS metrics to evaluate the performance of the TEEVE system at UIUC. The goal is to evaluate the performance of TCP, DCCP, and UDP over bundle of streams framework. We choose these protocols as they are the most common choices for wide area real-time transmission. Next, we discuss the results as follows:

**Experiment Set 1: Non Weighted Bundle Metrics**

In this set of experiments, non-weighted QoS metrics are used. Two bundles, one each from CWI and UCB are received at UIUC, and the number of streams in each bundle are varied between 2 and 3 streams. We performed the tests 6 times over different days and time, and the results show representative values of all of these tests. Due to space limitation, we show the results over CWI to UIUC connection, except for the bundle skew wherein skew between all three sites are shown. Figures 6.3, 6.4, 6.5, 6.6 shows the QoS achieved over TCP, DCCP, and UDP in terms of bundle delay, bundle jitter, bundle skew, and bundle throughput. The bundle delay over TCP is significantly higher (about 500%) as compared to DCCP and UDP. Similar trend is seen on the bundle jitter on TCP, with values ranging from 10 milliseconds to 2000 milliseconds. The jitter on DCCP and UDP is very nominal ranging from 0 milliseconds to 100 milliseconds. This shows that the aggressive congestion control of TCP has high negative impact on the timeliness of the protocol.

The figure 6.5 shows the bundle skew on TCP (ranging between 150 to 1500 milliseconds) as compared to the skew on DCCP and UDP. For the evaluation of bundle loss of information, a stringent deadline of 200 milliseconds (5 fps) is defined on macroframe receipt(frame rates below 4 fps lead to complete unusability of DIME sessions [8]). The results on bundle loss of information show that 87% of macroframes are dealt as lost in TCP due to high delays, while only 1.8%, 3.2% are lost in DCCP and UCP(though they are unreliable transport protocols). With a source frame rate of 12 macroframes/sec, the bundle throughput attained over TCP ranges between 0 to 80 KBytes/sec, and between 20 to 180 Kbytes/sec over DCCP and UDP.
Figure 6.3: Performance of TCP, DCCP, and UDP over Bundle Delay

Figure 6.4: Performance of TCP, DCCP, and UDP over Bundle Jitter
Figure 6.5: Performance of TCP, DCCP, and UDP over Bundle Skew

Figure 6.6: Performance of TCP, DCCP, and UDP over Bundle Throughput
**Experiment Set 2: Weighted Bundle Metrics**

This set comprises of evaluations done by using weighted metrics. Two to three camera streams constitute a bundle at each site, wherein in two streams case, top camera is given higher priority against bottom camera. In 3 streams case, 1 top camera and corresponding 1 bottom camera are given higher priority against third bottom camera. We also assign different rates to each camera stream depending on priority (15 fps, 10 fps, and 5 fps) to replicate the effects of variable macroframe sizes on the evaluation results. Figures 6.7, 6.8, 6.9 cover the obtained results.

![Graph](image)

(a) Weighted Bundle Delay

**Figure 6.7: Performance of TCP, DCCP, and UDP over Weighted Bundle Delay**

The weighted bundle delay on TCP shows significant reduction in delay as compared to the non-weighted bundle delay. Thus, the results indicate that one of DCCP/UDP can be used for wide area communication in DIMEs while TCP is prone to incur low QoS performance.
Figure 6.8: Performance of TCP, DCCP, and UDP over Weighted Bundle Jitter

Figure 6.9: Performance of TCP, DCCP, and UDP over Weighted Bundle Skew
6.2 SAS Kernel Implementation Details

SAS Dev Guide: SAS Kernel is implemented for our TEEVE system at UIUC. This chapter is divided into a handful of sections. They are:

1. High Level Overview
2. Getting Hands Dirty
3. Developing Guidelines

6.2.1 High Level Overview

SAS Kernel is a user level distributed kernel that enables remote users to interact in a 3D virtual space. The figure 6.10 gives an overview. In a nutshell, trigger is used for time synchronization of cameras, INTI is the camera code, XML is the configuration file, SASI is the wrapper around the end-device which connects to SAS Kernel. SAS Kernel is for networking and session control, and Emineo is the renderer code.

For deeper understanding of internals of SAS Kernel, more detailed system architecture is covered in the SAS Kernel Chapter 5.

6.2.2 Getting Hands Dirty

**Build from Source:** The latest SAS Kernel source file can be downloaded from the teeve svn: https://teeve-svn.cs.uiuc.edu/teeve/trunk/athena SAS Kernel is a licensed software and only members from Monet group can directly download from the above svn link. For outside downloads, please contact Prof. Klara Nahrstedt at UIUC to obtain proper license.

Once the source files are downloaded follow these instructions to build the software:

1. Run ./configure in the source directory.
2. Run make in the source directory
3. Run make install in the source directory

**Step-by-Step Guide to Running the System:** To run a complete TEEVE setup, users need to download and install appropriate INTI, Emineo, and trigger softwares at camera, renderer,
and trigger machines. More information on downloading and installing these softwares can be found at [TEEVEInstall: https://teeve-svn.cs.uic.edu/trac/wiki/TeeveInstall] and [TEEVERunning: https://teeve-svn.cs.uic.edu/trac/wiki/TeeveRunning].

Note: SAS Kernel is supported over both Linux and Windows machine. For end-devices (cameras, renderers, microphones) running on Windows should use the windows version and linux users otherwise.

![Figure 6.10: SAS Kernel Testbed Setup](image)

As shown in 6.10, XML based configuration file is used at the end-device to specify the stream and packet format and also the type, subtype of the sensors. The functions requested on per packet/stream can also be marked using this configuration file. Please refer to Figure 5.5 which shows a typical XML configuration. This XML configuration is read by the Wrapper entity belonging to SAS Kernel placed on the same end-device machine. The Wrapper initiates appropriate data and control connections with the SAS Kernel running at gateway server. Based on the chosen connection type, SAS Kernel at the gateway server starts TCP, DCCP, or UDP connections to receive the data streams from the Wrapper. The Wrapper then connects to the end-device on loop
back address and starts transmitting the streams to the gateway server.

The SAS Kernel software (SAS Kernel) in the gateway server, performs session initiation, and creates instreams to receive the streams. It also starts bundle managers which based on the correlations perform several bundle based operations like multi-stream synchronization, correlated congestion control on top of the bundles. The session manager bookmarks all the session information like comprising bundles, streams, and resources used in a given session. On receiving a request from the output device, the stream manager spawns a new thread to transfer each instream as outstream to the output device. To apply the user-requested functions on bundles, instreams, and outstreams, a transformation kernel (Opkernel) is used which provides a framework to load the requested functions (stored separately as dlls) at runtime. This transformation kernel then applies these functions to the bundles, instreams, and outstreams. The processed streams are then transferred to the output devices.

In case of multi-site TEEVE setup, multiple gateways (one per site) are used to transfer the streams to each of the participating sites. Given high bandwidth demand of 3DTI streams, efficient overlay topology formation becomes important to scale the number of sites. Thus, SAS Kernel provide an entity called Session Controller, a centralized software which controls the overlay topology formation between the gateways from each site. Currently, our Session Controller implements Mesh routing scheme between the sites. The session Controller is run on a node which can be universally connected from the gateways at all the sites.

A step-by-step guide to run the complete system with one camera and one renderer is as follows:

1. At trigger machine, cd pkg/, and run ./trigger.exe –num 1 (where 1 specifies the number of cameras going to be used)
2. At camera machine, appropriately configure XML file with correct DevicePort (same as the port which the end-device uses for data communication), Type, and Subtype of streams.
3. At gateway machine, cd athena/, and run ./gateway –port 1004 (wherein, –port specifies the port number on which the gateway listens for requests)
4. At camera machine, cd athena/, and run ./wrapper config_producer.xml
5. At camera machine, cd pkg/, and run INTI_trigger.exe -t 128.174.247.14 -r 127.0.0.1 (wherein, -t specifies the ip of trigger machine and -r specifies the ip of the machine running input
wrapper for this camera) –With this, the camera streams start to deliver at the gateway

6. At renderer machine, cd pkg/Emineo_gateway/bin, and run ./Vruilauncher -vislet Emineo -gs -o 3004 (where 3004 is the port on which it listens for the camera stream)

7. At renderer machine, cd athena/, and run ./wrapper config_consumer.xml –With this, camera streams should start displaying on the 3D display (you might need to zoom in by scrolling the mouse central button to see the video)

These steps give a detailed description of how to run the system. For running with multiple cameras, change of the –num parameter to number of cameras used, and repetition of the steps of starting the cameras at each of the camera machines is required. The only important configuration detail is that for multi-camera case, the config XML file should hold unique device id for each camera.

In case of multiple sites, start session controller by running cd athena, ./session_controller -p 9000 (where -p specifies the listen port) at any machine accessible via Internet from all the sites. Repeat the above steps on each site and enjoy real-time collaborative activities across sites!

6.3 Evaluation of SAS Kernel

We implement SAS Kernel for a real 3D Teleimmersion (3DTI) System, TEEVE at University of Illinois, Urbana Champaign. Our SAS Kernel is supported over both Linux and Windows operating systems. We evaluate the performance of SAS Kernel in TEEVE.

6.3.1 Experimental Setup

TEEVE typically includes 4 to 5 sites, each producing 4 to 5 streams comprising of 3D cameras and microphones. Each site spawns a gateway for the SAS Kernel. Each stereo camera produces a variable 3D video stream ranging between 6 to 10Mbps. To evaluate the strength of the SAS framework, third-party softwares for 3D camera from UC, Berkeley and renderer from UC, Davis are used. No source code modification in these third party softwares was needed and these end-devices could easily interface with SAS Kernel by only specifying device configuration in simple XML file.
We compare the performance of SAS Kernel in terms of overheads incurred on a) End-to-End Delay, b) CPU, and c) Bandwidth. We perform two experiments: Experiment 1) SAS Kernel consists of one gateway with 1 to 6 bundles with 2 video streams each, Experiment 2) SAS Kernel consists of two gateways, each with 1 to 6 bundles with 4 video streams each, and 2 output devices (renderer and the other gateway). So, each gateway receives total 24 instreams and sends 24 corresponding outstreams. It is to be noted that 12 and 24 streams per site is a large workload in DIME scenario in terms of bandwidth (120 to 240 Mbps), CPU (24 to 48 threads), and current applications. Thus, the evaluation highly stresses the system. Moreover, multiple gateways can be spawned to balance the load in the event of dramatic increase in streams. For repeatability, we use a recorded creative dance performance. For the gateway server, we use 4 Dell Precision 670 with dual Intel Xeon processor.

6.3.2 End-to-End Delay Overhead

The major goal of the kernel is to support real-time streaming even under heavy loads. We evaluate the total delay overhead added by the SAS Kernel wherein total delay is the difference between the entry time of a frame and the exit time of that frame from the SAS Kernel. Figures 6.11(a), 6.11(b) show that the total delay is less than 3 milliseconds even for 12 concurrent streams, and increases minimally on using 2 sites and 24 streams. This shows that SAS Kernel meets the soft real-time requirements of streaming while efficiently providing SAS.

6.3.3 CPU Overhead

It is important for SAS Kernel to scale in terms of CPU demands as large number of end-devices is added to the system. As shown in Figures 6.11(c), 6.11(d), the average CPU overhead ranges between 2% for 2 streams to 20% for 12 streams in Experiment 1. On doubling the number of sites and streams, the average CPU requirement only increases to 10% for 4 streams and 30% for 24 streams. This emphasizes that SAS Kernel demands low CPU even when large number of sensors are connected to it.
Figure 6.11: SAS Kernel Evaluation
6.3.4 Bandwidth Overhead

SAS Kernel adds S-RTP header on the data packets and uses Google Protobufs for marshalling S-RTP frames. Some DIME applications are bandwidth hungry, so it is important that SAS Kernel itself does not add too much bandwidth overhead. For the current implementation of the SAS Kernel, only a fixed cost of 22 bytes per frame is incurred as S-RTP header. The Google Protobuf only adds 4 bytes to the header. Thus, a total of 26 extra bytes per frame over frame size ranging from 2KB to 30KB for 3D-video frames and 140Bytes of audio frames is incurred.
CHAPTER 7

CONCLUSION

Our main thesis is that correlated multi-streaming is an important requirement in emerging cutting-edge multimedia technologies like DIMEs. The contributions of our work are manifold:

**Contributions:** We present a new concept called “Bundle of Streams”, along with a suite of QoS metrics to characterize and evaluate the emerging distributed interactive multimedia environments. The Bundle of Streams model captures the important spatio-temporal correlations among the multi-modal sensory streams in DIMEs, and serves as the conceptual foundation for a new set of QoS metrics such as Bundle Delay, Bundle Jitter, and Bundle Loss of Information. We provide formal definitions of these concepts and metrics, and validate them by comparing the performance of three transport layer protocols in a real DIME system.

We also propose that multi-streaming in DIMEs should be modeled as a real-time, generic, flexible, scalable, and robust Streaming as a Service (SAS) for highly correlated sensory data over the Internet. We introduce the concept of SAS and its implementation in the distributed SAS Kernel. SAS Kernel is a proof-of-concept architecture of this SAS model. SAS Kernel supports various types of sensors, transport and session protocols, as well as dynamically loaded functions such as congestion control, compression, and synchronization. Our experiments in a real DIME testbed indicate that SAS Kernel is successful at providing the service without much overhead time-wise (i.e., delay) and space-wise (i.e., bandwidth).

**Lessons Learned:** The key lesson we learned in this research space is that presence of high correlations among sensor streams are important properties to take into account in the upcoming interactive multimedia technologies. Exploitation of these correlations to enhance the quality and user experience while optimizing on the resources is an important research avenue in these highly immersive environments. Our proposed Bundle of Streams concept and Bundle QoS Model is a first step to define and measure QoS across correlated streams.
We learned that traditional QoS metrics on per stream basis fail to accurately measure the QoS across sets of correlated streams and that our Bundle QoS Model is needed to accurately measure QoS over these streams. We also learned that in many scenarios Weighted Bundle QoS Metrics are more relevant as compared to Non-Weighted QoS Metrics to focus on only a subset of higher priority sensors. We also faced challenges in measuring end-to-end QoS metrics across distributed sites due to distributed clock synchronization problems and the accuracy of the Bundle QoS metrics will depend upon the accuracy of the distributed clocks or other synchronization methods used. This challenge holds for any QoS metric whether traditional or Bundle QoS metrics.

Another important lesson relates to building the Bundle based SAS Kernel which re-emphasizes the gain in terms of QoS and QoE performance we achieved by ensuring soft real-time correlated multi-streaming service. Over the years of building streaming services for 3D Tele-immersive systems comprising of varied sensory end-devices, we came across an absolute need to build an end-device agnostic streaming service such that the service need not be rewritten to support different end-device specifications and could easily connect to any streaming end-device. With this goal in mind, a unified XML based SAS Interface is included in the SAS Kernel design. This decision worked very well and we could support easy configurability across a large number of new sensors and devices. As the SAS Kernel is being distributed to other research groups and being used for multiple applications, we are sure to find more areas of improvement and practical lessons relating to it.

**Future Directions:** Several researchers from our group are already adopting the Bundle of Streams concept and interesting correlations between Bundle Quality of Service (QoS) and user perceived Quality of Experience (QoE) are being derived, Bundle of Streams based real-time OS Kernel is being proposed, and Bundle of Streams based synchronization schemes are being designed (many papers are currently in submission in the top conference in multimedia).

As future directions, we want to focus on extending the current set of Bundle of Streams Metrics to incorporate more complex and domain specific metrics which can be applied to measure other conglomerate properties across correlated streams. In the space of SAS Kernel, we want to extend the current implementation to support more runtime functions on the bundles and streams like bundle congestion control, bundle bandwidth management, bundle synchronization, and bundle scheduling on CPU as well as over network.
REFERENCES


[12] X. Qin, “A delayed consistency model for distributed interactive systems with real-time con-


