STREAMOS: DISTRIBUTED OPERATING SYSTEM FOR CORRELATED MULTI-MODAL STREAMING APPLICATIONS

BY

RAOUL VICENTE RIVAS TOLEDANO

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Doctoral Committee:

Professor Klara Narhstedt, Chair
Professor Roy H. Campbell,
Associate Professor Indranil Gupta,
Professor Dongyan Xu, Purdue University

Abstract

3D Teleimmersive Systems (3D TI System) are geographically distributed systems that enable remote collaborative activities. 3D TI Systems are composed of cameras, sensors and microphones to capture the video, audio and sensory data from each site at different geographical locations. These data streams are then disseminated to the local and remote sites. At each remote site the audio, video and sensory information is aggregated with the information from the local site and rendered at the displays, haptic devices and speakers to recreate a fusion of the remote and local spaces where local and remote participants can interact and participate in collaborative activities. Resource and device management in 3D TI Systems poses several challenges: Large-Scale of Distributed I/O Devices, Time and Space Correlated Streams, Diversity of Interactive Activities and Non-Standard Heterogeneous Multimodal Interfaces. These challenges leave a significant burden to the user-space application implementing the 3D TI System. Despite this high-burden, resource and device management in 3D TI Systems is implemented as a user-space application. The main reason is that current approaches to resource and device management do not consider all the challenges in 3D TI systems. In this dissertation, we address the problem of designing Distributed Operating System services that address the device and resource management challenges in 3D TI. Our Distributed Operating System, StreamOS, is composed of 6 layered subsystems:

1. Cyberphysical Activity Layer provides a model for activities and stream processing in 3D TI Systems.

2. Kratos: An Activity Management and Detection Layer provides an activity detection system based on a Support Vector Machine for 3D TI System metadata. This layer provides an initial basis to address some of the challenges that arise due to the Diversity of Interactive Activities by providing activity information to all other layers of StreamOS.

3. Decima: A Device I/O Management Layer addresses the device management challenges that arise from Non-Standard Heterogeneous I/O devices being plugged and unplugged due to the Diversity of Interactive Activities. It also addresses the rapid changes in hardware of these Non-Standard Heterogeneous I/O. This layer addresses the problem of providing contextual

support in terms of location and identification for time and space correlated groups of interactive streams that arise from the Distributed Nature of I/O devices.

4. Prometheus: A Streaming as a Service Layer that provides end-to-end data delivery for I/O streaming devices. This layer addresses the challenges that arise from Distributed Correlated Streaming I/O devices by providing streaming of Bundle of Streams across geographically distributed TI sites. It also solves the challenge of interfacing with Non-Standard Heterogeneous I/O devices by providing a universal interface that can be accessed by a large range of multimodal devices. Finally, this layer also provides user-defined processing functions to Bundle of Streams to address the challenges in terms of activity-driven stream processing due to the Diversity of Interactive Activities and the time and space Correlated nature of Streaming I/O devices.

5. Zeus: A Real-time Stream Scheduler Layer that provides CPU Quality of Service guarantees to groups of correlated interactive streams (i.e., Bundle of Streams). As part of this layer, we provide a Process Calculus for analysis of dependencies and concurrencies in time and space correlated streams, and novel algorithms that provide scheduling for concurrent and codependent streams based on multi-core Earliest Deadline First (EDF) policy. This layer addresses the dependencies at the CPU Task Scheduling level. The dependencies arise from Correlated Streaming I/O devices and from the variability in the demand of CPU resources that due to the Diversity of Interactive Activities.

6. Hera: An Activity Based QoS Estimator that provides offline estimation of the QoS requirements in terms of bandwidth and CPU utilization of the 3D TI System. As part of our solution, we provide a QoS Model that estimates the QoS Requirements of a 3D TI Session based on the activity and the type of devices used during the session. This layer addresses the contingency and variability of QoS requirements caused by the Diversity of Interactive Activities.

To my parents for their utmost support and love

Acknowledgments

First and foremost, I want to thank my advisor, Prof. Klara Nahrstedt, for all her guidance and support during my graduate studies. I want to thank her the opportunity of being part of her research group since I was an undergraduate student. Her advice and constructive criticism proven invaluable during the course of my research in the Teleimmersion group. I would also like to thank my thesis committee members, Prof. Roy H. Campbell, Prof. Indranil Gupta and Prof. Dongyan Xu for their insightful comments, advice and suggestions.

I want to thank Dr. Robert Kuhn for all his comments, advice and encouragement during the course of my research. Also, I am thankful for the internship opportunity in his group at Intel, this opportunity was a true learning experience.

Besides, I want to express my gratitude to my colleagues in the MONET research group. I want to especially thank Dr. Ahsan Arefin for his encouragement, and help throughout discussion and collaboration; Pooja Agarwal for her amazing collaboration in many of the research projects in the Teleimmersion group and Dr. Hoang Nguyen for his guidance during my first research project as an undergraduate student. I want to thank Pengye Xia, Aadhar Jain, Dr. Zixia Huang, Dr. Wanmin Wu, Dr. Shu Shi and Dr. Zhenyu Yang for all their assistance conducting experiments. I also want to thank other current and past members of the MONET research group; among others, Rehana Tabassum, Kurchi Hazra and Dr. Long Vu.

Also, I want to thank Dr. Cinda Heeren for introducing me to my advisor. Without her introduction, none of this research would have been possible. I also want to thank Prof. Marco Caccamo and Prof. Bill Sanders as I had the opportunity in collaborating with them in exciting research projects. I want to thank Prof. Samuel King and Prof. Joseph Torellas who provided me with great inspiration through their lectures.

I want to thank Lynette Lubben, Cheri Helregel, Andrea Whitesell, Jenny Applequist, Anda Ohlsson, Rhonda McElroy and Mary Beth Kelly for their help with all the administrative issues along this years. Also I want to thank Darwin Miller and Barbara Leisner for all their help solving all the technical issues and for helping us keep our lab running smoothly.

I want to thank my friends Thomas Vercillo, Joseph Miksan, Ryan Bach,

Brian Boyle, Nicole and Sasan Shabrou, Ryan Hill, Daniel Stich, Effren McKissick and Brent Hronik. They also helped along the way through fun, relaxing and memorable times, which are very important especially after deadlines and during weekends.

My dearest appreciation is to my parents for all their support along my entire education years. None of this would have been possible without them, and therefore, I dedicate my thesis to them. I also want to thank my siblings Daniel and Manuel for all their encouragement all these years. Also, without their encouragement this thesis would have not been possible. Finally I want to thank my extended family for their support and encouragement.

This material is based in part upon work supported by the National Science Foundation under Grants NSF CNS 07-20702, NSF IIS 08-40323, NSF CNS 08-34480 and NSF NETSE 10-12194 and through the Universal Parallel Computing Research Center (Intel-Microsoft).

Table of Contents

Table of Symbols

Symbol	Description
V	View of Interest
S_i	Stream i
$o(S_i)$	Interest Function
$Y^{SN^{PS}}$	Stream Differentiation Function
f_k	k_th frame of a Stream
P_{S_i}	Period of Stream S_i
BS_{PS}	Bundle of Streams in Physical Space PS
PS	Physical Space
$\text{cor}_{PS}(S_i, S_j)$	Correlation function between Streams S_i and S_j
SN^{PS}	Set of all Streams in Physical Space PS
S^{In}	Instream associated with Stream S
S^{Out}	Outstream associated with Stream S
$(M, \circ)$	Composition Monoid
F	Processing Function
G	Processing Function
Φ_S	Set of valid frames for Stream S
$\{F_i\}_{i=1}^{m}$	Sequence of Processing Functions
T_i	Task with identifier i
ST	Set of all processes in the system
$J_k^{T_i}$	Job k_{th} of Task T_i
D_i	Deadline of Task T_i
P_i	Period of Task T_i
E_i^k	Release Time of Job $J_k^{T_i}$
FF_i^k	Finish Time of Job $J_k^{T_i}$
G	Group of Processes
C_i	CPU-time units required by the process to execute the Job
Y_i	Priority of Task T_i
$T_i \rightarrow T_j$	Dependency Relationship: T_j depends on T_i
$T_i \mid T_j$	Concurrency Relationship: T_i concurrent with T_j
ϵ	Skew between two Streams
CG	Constraints Graph
V	Set of Vertices of Constraint Graph
$\overline{E}$	Set of undirected edges of Constraint Graph
$\overrightarrow{E}$	Set of directed edges of Constraint Graph
κ	CPU identifier

K	Set of CPU in the local Content Delivery Gateway (CDG)
R_κ	Residual budget of CPU κ
ST_κ	Process set for CPU κ
a_i	Admission decision for Task T_i
HP	Hyper-period of the Process Set in the CDG
Δ	Set of all absolute deadlines in the hyperperiod HP
A	Activity Set
a	Activity
VS	Visual Space
Dev	Set of Devices in the 3D TI System
q	Quality Parameter defining the level of QoS
C^{In}	CPU-time units required by the process one frame of the Instream
C^{Out}	CPU-time units required by the process one frame of the Outstream
BW^{In}	Bandwidth required to disseminate one frame of the Instream
BW^{Out}	Bandwidth required to disseminate one frame of the Outstream
$\tilde{S}$	Variable Rate Stream
$\{\Delta_k\}_{k=1}^{n}$	Sequence indicating the time interval between the frames of a Variable Rate Stream
t_k	Arrival timestamp of frame f_k
X_k	A Random variable associated with frame f_k
Ω_k^X	Sample space of Random Variable X_k
$\bar{X}_k$	Sample mean of Random Variable X_k
CDF	Cumulative Distribution Function
$\mathcal{N}(\mu, \sigma^2)$	Normal Distribution with mean μ and variance σ^2
Z_S	Largest frame size of Stream S
z_k	Size of frame f_k
S^D	Dominant Stream S
φ	Function used in the regression model
β	Parameters of a function used in a regression model
Π	Set of valid regression models
SSE	Sum of Squares Error
$\mathcal{T}$	Windows Size for Support Vector Machine
N	Number of metadata parameters in the feature vector χ
χ	Feature vector for Support Vector Machine
α_i	i^{th} class label for sample feature vector χ_i
γ	Mapping function between sample χ and class α
sgn()	Sign Function
w	Weight vector for sample feature vector χ
Θ	Constant threshold for mapping function γ
argmax()	Argument of the maximum function
SC	Shirt Color
BdS	Body Structure
H	Height of the participants in meters
W	Weight of the participants in kilograms

Chapter 1

Introduction

1.1 3-D Teleimmersion

3D Teleimmersive Systems (3D TI System) are geographically distributed systems that enable remote collaborative activities. 3D TI Systems are composed of cameras, sensors and microphones to capture the video, audio and sensory data from each site at different geographical locations. These data streams are then disseminated to the local and remote sites. At each remote site the audio, video and sensory information is aggregated with the information from the local site and rendered at the displays, haptic devices and speakers to recreate a fusion of the remote and local spaces where local and remote participants can interact and participate in collaborative activities.

These systems entail much richer media than traditional interactive audio video systems like Skype [1] and NetMeeting [2]. They represent a new class of Telepresence systems [3] [4] [5] with 3D multi-view video capabilities that aim towards full body coverage and new immersive interactions in the same virtual space. Some examples of this systems are Cave [6], TEEVE [7] and Coliseum [8]. These 3D TI Systems allow fast-paced real-time collaborative activities such as exergaming [9], distant-learning and tele-health [10].

Furthermore, these 3D TI Systems run activities where there is a teacher-student, doctor-patient or trainer-trainee relationship. The TI System is then controlled by the trainer, the doctor or the teacher to accomplish a certain goal. Figure 1.1 shows the concept of a 3D TI System in which a physiotherapy doctor is instructing 2 patients from different geographical locations how to perform rehabilitation exercises.

During the activity, the doctor or the trainer might change the view of system to be able to see the patient perform a prescribed exercise from a better angle [11]; the doctor, may also, instruct the patient to connect a sensor that monitor his heart rate or to use an accelerometer that measures how much force he is applying while performing the exercise. In a virtual fencing game, the participants will use virtual swords using smartphones [12] or prop swords that light and provide force-feedback as the users interact; also, the participants might connect headphones and microphones to communicate during game-play [9].

Figure 1.1: Example of physiotherapy application in a 3D TI System

In summary 3D TI Systems are characterized by: 1. Rich audio-visual interactions in the virtual space. 2. Large-scale of streaming devices. 3. Multi-Site Collaboration.

These characteristics create significant challenges in the implementation of a 3D TI System. In the following section we describe some of these challenges.

1.2 Challenges in 3D TI

1.2.1 Distributed Nature of I/O Devices

A 3D TI System is comprised of multiple geographically distributed sites connected through Internet2. Most of the activities in these systems are guided by a trainer with the purpose of achieving a certain goal. Hence, the trainer must be able to identify the location of each device based on the specific 3D TI site and yet control each 3D TI device independently of their location. This requirement imposes the challenge of providing heterogeneous access to reduce the complexity of interfacing with devices whether they are connected locally or remotely. More specifically it is important to provide Distributed Device Management that supports high-level platform independent access to the device in the 3D TI systems. It must also provide consistent resource naming to sites and devices in 3D TI Systems that allows to preserve the geographical location awareness and context of each group of devices and streams associated with each particular 3D TI site.

1.2.2 Correlated Streaming I/O devices

Each input device in a 3D TI System capture a different *Point of View* of the same environment at the same time, therefore the captured frames are *correlated in time and space*. For example, a 3D TI site might be composed of upper

3D cameras that capture the face of the participants and lower 3D cameras that capture the lower extremities of the participants. If these cameras are not properly synchronized, the resulting 3D model of the person might be disconnected and the movement will appear uncoordinated. Therefore, these streams are highly sensitive to inter-stream skew and they must be tightly synchronized before they are rendered at the output devices in the local and remote sites. Figure 1.2 shows an example of a 3D TI System with multiple cameras at different Point of Views. In this example each of the cameras capture a different angle of the same dancer in each of the TI sites at the same time, therefore the video streams from these cameras are correlated in time and space and their frames must be synchronized to minimize the skew among them before they are rendered at the displays in each site.

In order to address the challenge of Correlated Streaming it is necessary to provide support for Real-Time Stream-based Scheduling and Processing Run-Time Services that consider the dependencies between streams.

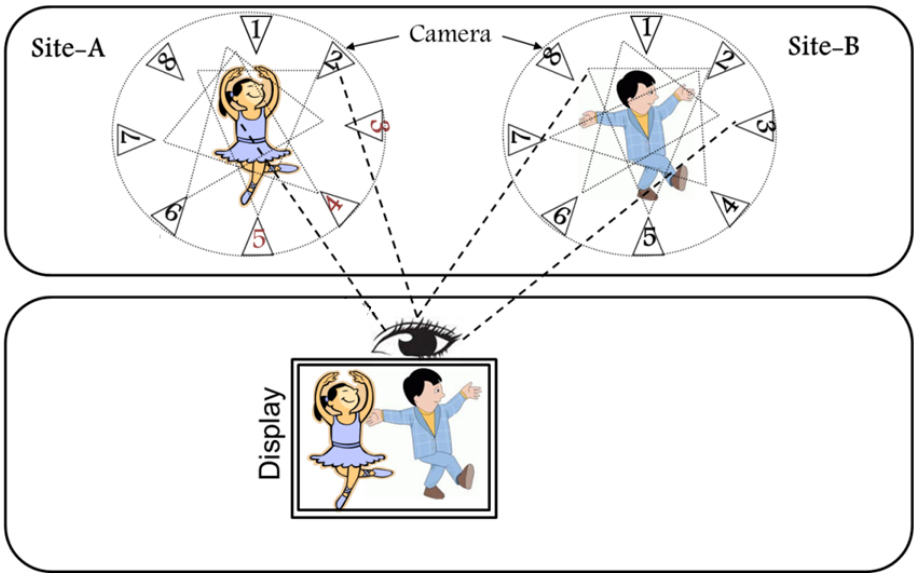


Figure 1.2: Example of Correlated Streaming in 3D TI

1.2.3 Diversity of Interactive Activities

3D TI Systems are characterized by interactive cyber-physical activities that range from videoconferencing and distant learning to exergaming and physiotherapy. This highly interactive activities impose tight QoS constraints in terms of delay, skew and jitter. If streams are delayed the participants will experience a severe lack of interaction as the physical movements of the participants in the local site will be uncoordinated with the movement from the participants in the remote site. Previous studies have shown that immersion in cyber-physical activities in 3DTI systems is severely impacted if the end-to-end delay across the

Activity	Device Type	# of Devices	QoS Constraints
3D Conferencing	Microphones, Cameras	Small	High Quality Audio Low Skew
Exergaming	Cameras, Sensors	Large	High Frame Rate Low Delay
Tele-Health	Cameras, Sensors and Microphones	Large	High Quality Video High Frame Rate Low Delay

Table 1.1: QoS constraints of various 3D TI activities

3D TI geographical sites is higher than 200 ms [13]. Low Jitter is also a significant factor in 3D TI systems as participants will experience lack of smoothness in motion movement and therefore this will also significantly degrade the immersive experience.

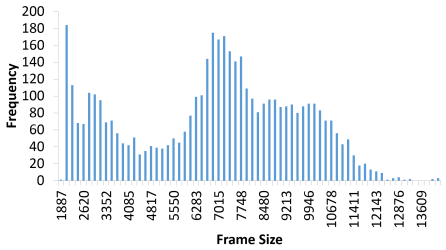
Moreover, 3D TI Systems are composed of a large number of input devices including 3 to 4 stereo cameras at each site, a microphone for each participant and additional sensors that include accelerometers, and haptci devices. To aggravate this issue of scalability, a typical stereo camera requires 1.5 Mbps of bandwidth for streaming data. The high bandwidth requirement and the large number of streaming devices make over-provisioning a prohibiting approach. Therefore, 3D TI systems require specialized resource management and reservation approaches that provide statistical QoS guarantees.

As additional complexity, activities drive the type and number of devices in a 3D TI System. For example, simple activities like 3D conversation require only a single camera and a microphone while other more complicated physical activities like exergaming might require multiple cameras to cover a much wider field of view required in these activities. Also, Telemedicine activities might require additional body sensors to capture fine grained movement or heart rate variations in the patients.

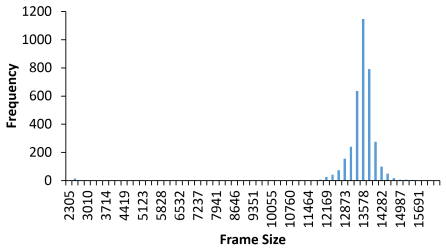
Additionally, different activities define different QoS parameters [14]. For example, 3D Conferencing requires high quality audio and low skew to achieve lip synchronization, whereas exergaming activities like virtual fencing [9] require high frame rate cameras and low delay to achieve high level of interactivity. Table 1.1 shows some of the QoS constraints of various 3D TI activities.

In addition to this, frames from 3D video cameras have different rate and bandwidth based on the complexity of the captured scene, as complex scenes require more processing time and a higher number of pixels to be represented. For example, the average frame size of a 3D video frame with only one person standing is 11 Kb, however, the average frame size of a 3D video frame with on person sitting is 5 Kb. Due to the variation in frame rate and frame size, the type of activity in the 3D TI System plays a key role in driving the QoS requirements. Figure 1.3 shows a histogram of the distribution of frame sizes and frame rates for the Walking and Standing activities in a 3D TI Systems.

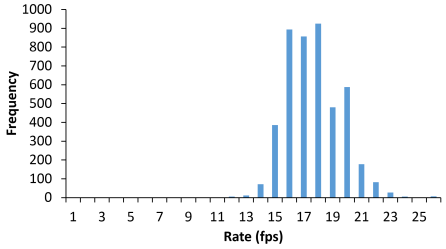
To address the challenges created by the diversity of interactive activities in 3D TI Systems, it is important to provide Time Sensitive Resource Allocators that provide real-time resource management and resource reservation based on the type of interactive activity in the 3D TI System. Additionally, it is important to provide Activity-driven Reconfigurable Device Management and Resource Allocation that allows devices to be enabled and disabled based on the need of the users and activities; and cope with the changes in the resource demand.



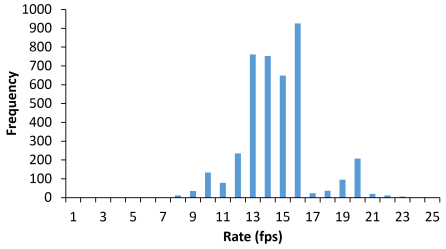
(a) Frame Size Histogram for Walking Activity



(b) Frame Size Histogram for Standing Activity



(c) Frame Rate Histogram for Walking Activity



(d) Frame Size Histogram for Standing Activity

Figure 1.3: Histogram of Frame Size and Frame Rates for different 3D TI activities

1.2.4 Non-Standard Heterogeneous I/O devices

Devices are heterogeneous and range from small factor sensors to 3D Stereo cameras. This heterogeneity poses the challenge of heterogeneous streams with different bandwidth and rate. Moreover, it creates the challenge of multimodal interfaces due the lack of standard streaming formats. For example, Bumblebee Stereo Cameras [15] and Kinect cameras both provide very different protocols and Application Programming Interfaces (APIs) used to control and interface between the device and the controlling entities (i.e., activity). Additionally, the rapid changes in device hardware revisions complicate implementation of software that uses these contingent non-standard interfaces to access the hardware. The rapid change of hardware in 3D cameras and sensors, the lack of standard stream formats and the heterogeneity of devices present the need of enabling Universal Access to large range of heterogeneous streaming devices.

1.3 Thesis Statement

3D TI Systems can be implemented as an application running on top of a general purpose operating system like Linux or Microsoft Windows. However, these general purpose operating systems do not provide any specialized device and resource management support for 3D TI Systems and leave a significant burden to the user-space application implementing the 3D TI System, as this applications must implement specialized device and resource management entities that satisfy the challenges of 3D TI systems. A doc implementation of device and resource management is a significant work and therefore it can lead to a significant barrier in the deployment of ubiquitous 3D TI systems.

Some 3D TI systems rely on special purpose run-time systems to partially alleviate the need of handcrafting and implementing all the special purpose algorithms required to support device and resource management in 3D TI. An example of such system is Coliseum [8]. Coliseum uses the Microsoft Windows Direct 3D API [16] for capturing and rendering audio and video and a streaming middleware called Nizza [17] for data processing and dissemination. Nizza provides support for processing and dissemination of streams. However, Nizza only considers data dissemination and processing and does not consider any other of the challenges of 3D TI Systems. Therefore, Coliseum still relies on a user-space application that implements specialized scheduling and device management in 3D TI Systems.

The main obstacle on eliminating the burden of implementing resource and device management as user-applications in 3D TI systems is that current approaches to resource and device management do not consider all the challenges in 3D TI. Interfacing technologies like Distributed COM [18] and CORBA [19] only provide Universal Access to a large range of interface, however, they do not provide any resource management needed to enforce policies to provide Quality of Service in 3D TI Systems. Throughput-Oriented Distributed Operating Systems such as Amoeba [20] and Chorus [21] address the distributed nature of 3D TI Systems and provide resource management policies but they do not provide Real-Time Scheduling necessary to achieve QoS guarantees in 3D TI Systems.

Real-Time CPU Scheduling approaches like Rialto [22], DSRT [23] and RTOS [24] provide QoS guarantees to processes, but their focus is only on the Local CPU Resource Management and therefore they are insufficient due to the distributed nature of 3D TI Systems. Real-Time extensions to Distributed Operating Systems, like MachRT [25] for the Mach Distributed Operating system, provide QoS support to existing Distributing Operating Systems. However, these approaches do not consider the dependencies created across multiple processes in correlated multi-stream 3D TI systems. These dependencies impose additional temporal constraints in terms of causality and concurrency to the traditional real-time scheduling in multiprocessor environments.

Because resource and device management in 3D TI Systems are highly driven

by the activity and the users in the system, current approaches do not address this Activity-specific Context Information. Having contextual information can significantly simplify the resource management in 3D TI by providing a higher level of semantics in terms of device and resource management.

In this dissertation, we address the problem of designing Distributed Operating System services that provide a holistic approach to resource and device management in 3D Teleimmersive Systems. We show that holistic resource and device management significantly simplifies resource and device management in 3D TI and therefore it reduces the burden to the application implementing the 3D TI System.

Hence, our thesis statement is that: *Activity-aware Real-Time Stream-based resource and device management is a necessary component in multi-party, multi-stream 3D Teleimmersive Systems*

1.4 Major Contributions

Our major contribution is to propose a holistic approach to resource and device management in which the activities, the users and the inherent characteristics of the 3D TI Systems require a coordinated set of policies and mechanisms shared across all the components in the 3D TI System in order to successfully provide QoS guarantees and adequate resource and device management. As part of our solution we introduce StreamOS, a Distributed Operating System that provides:

1. **Correlated Stream Soft Real-Time scheduling:** As part of our contribution we introduce a Process Calculus to model the relations of concurrency and dependency between time and space correlated streams in 3D TI Systems. This novel model provides powerful notation that allows modeling the complex constraints of groups of streams in 3D TI Systems. We use this model as a basis to design a novel scheduling algorithm for concurrent and codependent tasks in multi-processor systems based on the partitioned Earliest Deadline First algorithm [26]. Our novel scheduling algorithm uses a concurrency budget based on the laxity of the task (i.e., residual budget) to minimize the amount of the skew between tasks based on their actual running time. As part of our contribution we introduce an admission control for group of streams.

2. **Activity-based QoS Management for 3D TI Systems:** As part of our contribution, we introduce a QoS model for processing and dissemination of streams in 3D TI Systems in which the activity is the main driving factor of the QoS requirements in 3D TI Systems. Our model considers that the activity determines the type of devices and the type of stream processing required in 3D TI systems. We use this model to design a Support Vector Machine (SVM) to perform activity detection based on metadata. Also, we use this model to design an offline profiling

tool based on parametric estimation to obtain probabilistic bounds for the QoS requirements of a 3D TI session based on a particular activity.

3. **Universal Device-Stream Management and Interface:** As part of our contribution we introduce the Streaming as a Service concept in which real-time data streaming is provided between input and output devices as a transparent layer. In this model access to disseminating infrastructures is provided through a universal interface in which multimodal devices require no source code modification to interface and instead they provide a specification about their streaming protocols. Our paradigm introduces the concept of groups of time and space correlated streams (i.e., Bundle of Streams [27]) as first class objects. It also introduces hierarchical uniform naming for devices, sites and session in 3D TI Systems. This paradigm solves the problem of location-context preservation in 3D TI and Telepresence Systems as it allows to uniquely associate a device with a site and a session. We use this paradigm to design a streaming framework in which input and output devices stream data through networking tunnels without source modification. Our streaming framework enables location context-based real-time processing of user-defined functions over correlated streams. Finally, we also use this paradigm to allow location context-aware I/O management of multimodal devices to enable automation of I/O management in TI activities.

1.5 Dissertation Outline

The rest of the Dissertation is organized as follows: Chapter 2 discusses the Related Work relevant to StreamOS; Chapter 3 introduces our model and assumptions of a 3D TI System; Chapter 4 describes the Architecture Overview of StreamOS; Chapter 5 describes Zeus, a real-time scheduler for time and space correlated streams based on our novel Process Calculus for modeling dependencies in correlated streams; Chapter 6 describes Prometheus, a Streaming framework based on our novel paradigm Streaming as a Service; Chapter 7 describes the architecture of Decima, a Dynamic I/O management system for 3D TI Systems based on a dual-virtualization architecture; Chapter 8 describes Kratos an activity detection model for 3D TI Systems based on Support Vector Machines; Chapter 9 describes Hera, an offline profiling tool that estimates the QoS requirements for a particular 3D TI session based on our novel activity-driven device and process QoS model for 3D TI Systems; finally, we conclude in Chapter 10.

Chapter 2

Related Work

General purpose Operating System have very little support for stream management and processing. Due to the advent of Multimedia Applications, several of this Operating Systems have added certain support. For example, recent versions of Windows have added Multimedia QoS support through the Multimedia Class Scheduler Service (MCSS) [28] and some Linux distributions have added streaming support through GStreamer [29] and PulseAudio [30], the support is mostly oriented to a wide range of multimedia systems and therefore provides very little support for the large scale of devices and streams found in 3D Teleimmersion. Also, these extensions lack support for time and space correlated streams. More specialized approaches are discussed in this Chapter as we provide a review of the related work as it pertains to some of the mechanisms required when designing an Operating System for 3-D Teleimmersion. The relevant work to our thesis can be classified in 5 main areas: 1. Run-time and Distributed Systems. 2. CPU Scheduling, 3. Streaming Systems and Protocols, 4. Device I/O Management and 5. Activity Detection.

2.1 Run-time and Operating Systems

Throughput Oriented Distributed Operating Systems do not provide QoS guarantees and multimedia streaming applications running on top must rely on over-provisioning to achieve adequate Quality of Experience (QoE) to the users. Some of these approaches include Amoeba [20], Mach [31] and Chorus [21]. However, the large-scale nature of 3D Teleimmersion and the high bandwidth requirements cause over-provisioning approaches to become unrealistically expensive.

Distributed Run-time Systems like Stampede [32] provide stream support for developing multimedia applications. Stampede provides support for inter-stream synchronization and real-time guarantees to cluster-based multimedia applications through the Space-Time Abstraction [33], however, they do not provide specific activity driven QoS as required in 3D TI systems and also they do not provide support for resource naming of device and streams to preserve the location-context information of 3D TI devices. While 3D TI Systems can benefit from Stampede, this approach itself doe snot provide a complete solution

as it will still require significant design and implementation effort in the user application space.

The meta Operating System Gaia [34] provides contextual support for closed systems (e.g., rooms), however, their system is too general and it would be very difficult to provide I/O management for 3D TI Systems. Also, the middleware Carisma [35] provides contextual information support, however, the system is tailored towards mobile devices.

Also, related are Exokernel [36], an Operating Systems that provides extensibility through application-level functions and the run-time system R2 [37] that provides interfacing and interposition for run-time functions. StreamOS borrows concepts from these systems, however, these systems alone do not provide an adequate solution as they lack support for real-time guarantees neither they are distributed activity or stream centric. Also, in [38], resource containers are presented to provide resource management over processes and threads for network servers. CoreOS [39] is a Distributed Operating System based on resource containers to allow dynamic scalability and resource isolation in distributed computing. Compared to [38] and CoreOS, our approach is at a higher level of abstraction spanning across sites, sessions and streams.

2.2 CPU Scheduling

The area of CPU scheduling can be divided in two main areas: 1. Throughput Oriented System and 2. Real-Time Systems

Gang scheduling [40] and co-scheduling address the problem of scheduling of multiple tasks with concurrent constraints. These approaches have been well studied for best-effort, throughput-oriented multiprocessing like those proposed by Ousterhout et al. [41], Feitelson et al. [42] and Frachtenberg et al. [43]. However, they are not suitable for periodic soft real-time environments with tight QoS requirements like those found in 3D Teleimmersion.

There have been many approaches to support real-time guarantees in commodity hardware (e.g., Rialto [22], DSRT [23], RTOS [24], RTLinux [44], RK [45]). These approaches, however, are centralized and are designed for independent tasks and therefore it is very difficult to map the QoS constraints from streams into tasks. A relevant approach is MachRT [25] that provides Real-Time extensions to the Mach Distributed Operating system. However, this approach do not consider the dependencies created across multiple processes in correlated multi-stream systems. Stream Dependencies are very important as they impose additional temporal constraints in terms of causality and concurrency to the traditional real-time scheduling in multiprocessor environments. Another important approach is iDSRT [46] in which the authors consider dependencies across multiple nodes for the critical infrastructure, however their approach is tailored towards small sensors in Wireless Local Area Networks and not towards large scale of streaming devices as required by 3D TI Systems. Another impor-

tant approach is GraceOS [47] in which the authors consider the power usage as another constraint in the system. Some other approaches also related to our work is the adaptive scheduling for legacy realtime applications [48] and the multimedia scheduler SMART [49].

In the recent years, there has been some interest in analyzing concurrent constraints for real-time systems. The most notable approach is Gang EDF [50], in this work, Kato et al. propose a global EDF algorithm for multiprocessors in which they apply EDF policy to gang scheduling schemes. However, their algorithms are designed for sporadic parallel tasks and not for periodic tasks as those found in multimedia streams. Also, Yuan et al. [51] proposed the use of a process control block to manage adaptation in codependent tasks. However, his work is more related to the adaptation of the codependent tasks. More recent is the work of Lakshmanan et al. [52], in which he proposes to bundle processes that access shared resources to reduce synchronization penalties and jitter.

Finally, related to our work is the Pi-Calculus from Milner et al. [53]. In his work, Milner specifies a Process Calculus to describe interaction between parallel processes. However, this calculus is very generic and it will be complicated to express the dependencies and constraints of correlated streams in compact manner. Therefore, while it can be used to describe such relationships it can become cumbersome to use them in 3D TI Systems with large number of streams. Another important approach is the ISO standard LOTOS [54], LOTOS is a Process Calculus to specify process ordering. This calculus specifies both concurrency and dependencies but models processes as gates and therefore it can also become complicated to specify the dependencies of the much higher-level streams in terms of LOTOS' processes.

2.3 Streaming Systems and Protocols

Several architectures for streaming gateways interconnecting Local Area Networks (LAN) through Ethernet, Bluetooth or wireless to provide access to the Internet exist in the literature. These approaches [55], [56], [57], [58], [59] are general architectures for home, sensor, and streaming gateways. However, these service gateways are limited to relaying, multiplexing, translating, and managing local resources only, which fail to satisfy the requirements of 3D TI Systems. There are some proprietary gateways developed for 3D Teleimmersive Systems, such as, HP Halo [4], Cisco Telepresence [3], and Technicolor, however they are tailored to cater closed applications and their internal functionality is publicly unknown.

OSGi [60] is a java-based service platform for home networks allowing service providers to dynamically load and deliver services to the end users. However, it is very cumbersome to build complex systems like multi-correlated streaming over low-level OSGi [61]. In [62], user-configuration is used for setting class of service policies in routers. These approaches are not designed for 3D TI Systems

and therefore they do not consider the correlated multi-streaming nature neither they consider the QoS requirements in 3D TI.

The streaming middleware Nizza [17] provides support for processing and dissemination of streams. However, Nizza only considers data dissemination and processing. It is used by the Coliseum [8] Teleprescence System as underlying streaming framework. However, Coliseum still relies on a user-space application that implements specialized scheduling and device management in 3D TI Systems. Therefore, it is unsuitable in itself as a solution to the resource and device management in 3D TI Systems.

CoolStreaming [63] is a Peer-to-Peer (P2P) distributed system for live media streaming. It provides bounded delay, which makes it suitable for live streaming. Another related approach is mTreeBone [64]. These approaches lack processing capabilities and they also lack support for time and space correlated streaming required in 3D Teleimmersion.

DASH [65], defined in the ISO/IEC 23009-1 standard, is a protocol for Streaming MPEG over HTTP. It provides dynamic adaptation, however it is only limited to MPEG video and it is unsuitable for the sensory and multimodal devices of 3D Teleimmersion. Similarly, RTCWeb [66], submitted as a draft to IETF, is a protocol that enables Real-Time Streaming of data and enables rich-media content in web browser through HTML5. This approach provides protocols that can be used for real-time streaming however it does not provide device management and policy enforcement to achieve QoS in 3D TI Systems, also the protocol does not consider the problem of preserving location-context of an stream in 3D TI Sites.

The CLUE data model (Controlling Multiple Streams for Telepresence) [67] proposes various Internet protocols and standards for interoperability of Telepresence Systems. This standard addresses some of the issues in terms of preserving the location-context of a stream and providing universal access across different interfaces. However, it only considers audio and video and it does not consider multimodal sensors. It also, does not consider mechanisms to provide Quality of Service, neither it considers diverse physical activities as required by more advanced 3D TI Systems.

System-S [68] is a near *real-time stream processing framework*, mainly used for data mining and analytics. However, the time constraints for data analytics are in the order of seconds and streams do not have the tight skew and jitter constraints of 3D Teleimmersion. This is similar to the case of *Value-Added Content Delivery Networks* like Conviva's Video Control Plane [69], Akamai HD Streaming [70] and Microsoft SmoothStreaming [71] that provide processing and adaptation for Video on Demand (VoD) streaming. They are also insufficient as delay is not significantly important in these scenarios where users can experience initial video delays of 3 to 5 seconds due to initial buffering.

Pipeline-based multimedia frameworks like GStreamer [29], DirectShow [16] and QuickTime [72] provide low-latency stream processing of audio and video,

however, they provide limited support for correlated multi-streams and also very limited support for multi-site support. Also, these systems have a fixed set of formats and it is very difficult to extend the system beyond these formats. Some of these architectures are somewhat extensible through the support of plug-ins, however, the API provided is tightly coupled with the formats supported and hence it is inadequate for 3D TI Systems where the instability of stream formats is significant.

RemoteDMA used in Infiniband [73], is a technique for data transmission over the network mainly implemented in hardware. Infiniband provides universal access to devices in a similar manner to iSCSI [74], however RemoteDMA is limited as the receiving device is not notified of the completion of the transmission request. While RemoteDMA can be used to improve the performance of 3D TI System, this paradigm alone is insufficient for 3D TI Systems that require synchronization to minimize skew.

2.4 Device I/O Management

In the area of I/O systems there has been a significant amount of work to enable dynamic hot-plug I/O systems. The Universal Serial Bus (USB) [75] architecture provides enumeration and universal access to devices through Device Classes. Similar approaches are [76] and and SCSI [77]. These device management systems work at a low level and access to the device is performed using Application Programming Interface (API) provided by the device driver and existing Operating System services like Video4Linux and Windows Media using standard multimedia formats. However, this is insufficient for TIs as the lack of standard formats in sensors and 3D cameras and the multi-modal interfaces of these devices will leave a high burden to the developer of the 3D TI System application

In the virtualization domain, Xen [78] uses device virtualization and a split device driver model to ensure isolation by providing virtualized access to devices from non privileged virtual machines. Also, VMWare Workstation [79] uses device virtualization and virtualization of the USB controllers (OHCI and EHCI) to enable virtualized access to USB devices, but in a local node and not in a distributed system.

Application level distributed I/O systems allow high-level multiplexed access to devices: Server Message Block (SMB/CIFS) [80] provides access to shared files and printers in Windows systems. PulseAudio [30] provides a distributed audio server connecting audio devices. iSCSI [74] is a distributed I/O system that allows remote access to SCSI storage devices. Network File System (NFS) [81] allows access to remote files through a Remote Procedure Call (RPC). However, existing application level distributed I/O systems do not provide adequate semantic support for groups of streams needed in TIs, neither they support multi-modal interfaces. Also, somewhat related to our work is the Virtual File

System (VFS) [82] layer of Linux. However, VFS uses files as first class objects, and therefore, the interface is not sufficient to leverage the virtual layer of VFS to provide universal access to a large class of stream-based devices.

An important approach for distributed systems is Uniform I/O System Interface for Distributed Systems (UIO) [83]. UIO provides a uniform interface for I/O in distributed systems, however it provides a very low level interface and tries to solve synchronization and atomicity problems found in concurrent I/O access. The Universal Middleware Bridge (UMB) [84] uses the concept of a universal interface for distributed I/O, however they do not virtualize the device and they instead publish the device interface. This approach will not scale properly in TIs, where the number of devices is large. Room-Bridge also aims at providing a universal interface, however, they use a simple fixed interface to write and read commands to devices. Room-Bridge is tailored for very simple devices like home automation systems and it would not work well in TIs with a large number of complex devices. Similar to these approaches is DomoNet [85]. DomoNet proposes to use a language, based on XML, to communicate with the devices. A translator interprets this language and forwards commands to the device. The OSGI [61] gateway also provides a common universal access for interfaces but the API provided is very low level and designed for much simpler interfaces and it would be unsuitable for streaming devices like those found in 3D TI Systems.

Somewhat related to our work is the Remote I/O (RIO) [86]. RIO uses an MPI-IO interface to communicate with devices. However, their approaches are orthogonal to our approach and they focus on minimizing the communication overhead rather than providing universal access. Also, somewhat related is MultiSense [87], a remote I/O control system for metropolitan cameras. MultiSense provides distributed I/O control of multimedia systems, however, their focus is mainly on scheduling and multiplexing of the rotation mechanism of the camera device.

2.5 Activity Detection

In [88], Niu et al. propose to use several linear motion sensors and aggregate the obtained sensory information to detect 3-D motion patterns associated with specific activities. The problem of this approach is that the space must be augmented with additional sensors in specific locations which might not be available in all 3D TI scenarios. Also, this approach is tailored towards detecting security breaches and therefore it is unsuitable for QoS estimation in 3D TI Systems.

Sung et al. [89] propose to use Kinect data to provide activity detection. This approach can benefit 3D TI Systems, however the lack of standards in 3D TI limits their practical use as different 3D TI Systems are implemented with different 3D video formats with little interoperability. This similar problem

occurs with other approaches that propose to combine sensory data with video, some of this approaches are [90] and [91].

Chapter 3

3D TI Model and Assumptions

3.1 3DTI Architecture

3D TI Systems are comprised of multiple geographically-distributed sites interconnected through the Internet2 network infrastructure. Each of these sites is composed of:

1. Input devices: At each physical site, multiple 3-D cameras, microphones and sensors capture the cyber-physical information of the local user in the 3D TI System.

2. Disseminating Infrastructure: Multiple streams from each local site are aggregated and processed at the local Content Delivery Gateway (CDG) and then disseminated to the Content Delivery Gateways at the remote sites.

3. Output devices: At each local and remote site, multiple displays, speakers and actuators render the multi-modal cyber-physical streams from the local and remote sites into a joint virtual environment.

Figure 3.1 shows a 3D TI System with 3 sites showing 3-D cameras (C), displays (R) and audio devices (A).

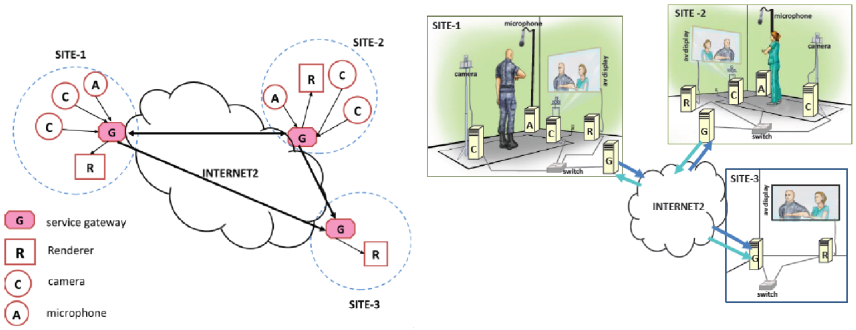


Figure 3.1: Example of a 3D TI System showing 3 sites

In a 3D TI System, temporally and spatially correlated streams form a stream group, referred to as the Bundle of Streams [27]. Streams forming a

bundle are processed and disseminated together from the local site to their destination in their remote sites.

The functional model of a 3D TI System can be decomposed in 3 logical tiers:

1. Activity Tier: representing the activity performed by the users of the 3D TI System (e.g., tele-health, remote learning)
2. Service Tier: representing the data streaming paradigm.
3. Physical Tier: representing the underlying physical devices of the system.

Figure 3.2 shows the functional model of a 3D TI System and its 3 main tiers.

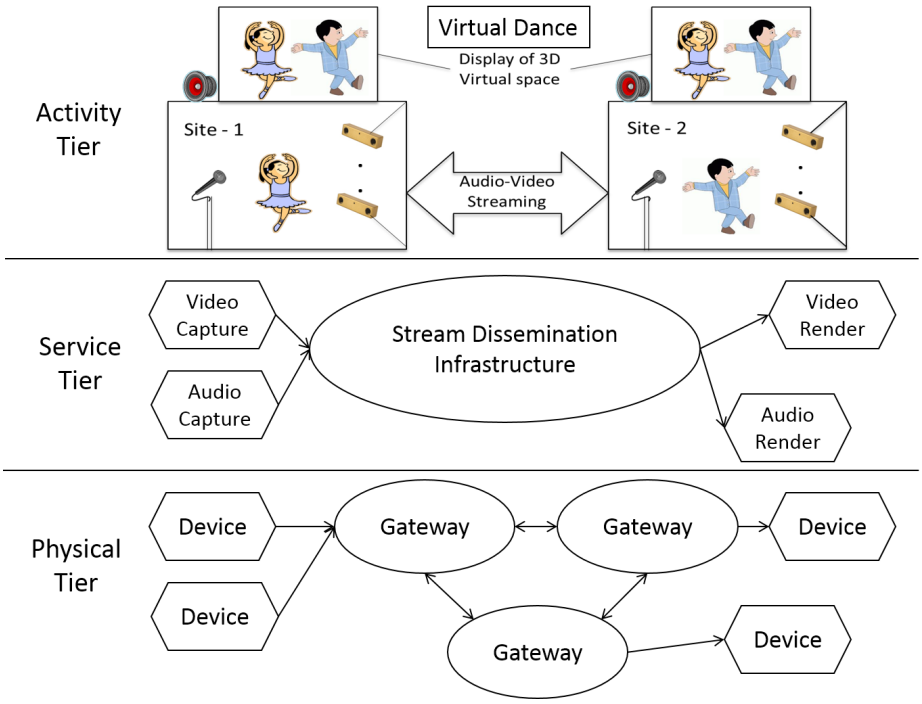


Figure 3.2: Functional Model of a 3D TI System

Activity Tier

Activities in 3D TI Systems are varied and range from simple 3D conversation to exergaming or remote telemedicine [92]. The contingency of activities in 3D TI Systems are critical as the nature of the activity drives the type and number of devices used in the 3D TI System. For example, simple activities like 3D conversation require only a single camera and a microphone while other more complicated physical activities like exergaming might require multiple cameras to cover a much wider field of view required in these activities.

Also, Telemedicine activities might require additional body sensors to capture fine grained movement or heart rate variations in the patients.

Activities in 3D TI Systems can be divided in two types:

1. Symmetric: In these activities each site controls each device and decides what to send to other sites. This type of activities includes videoconferencing (e.g., Skype) and distributed exergaming(e.g., multi-player online gaming).

2. Asymmetric: In these activities all the devices in the activity are controlled by one site. These activities are characterized by a fiduciary relationship in which one of the users drives the activity to achieve a specific goal (e.g., doctor-patients, trainer-trainee, teacher-student). The user that drives this type of activities requires remote control access to the devices at each site. For example, a doctor prescribing physiotherapy exercises has a control (e.g., Wiimote [11]) that allows him to change the view angle of the cameras to check if the patients are properly executing the exercises.

Service Tier

Data dissemination is an important component of 3D TI Systems, in which correlated data streams from the input devices at the local site are delivered over Internet2 to the remote sites. In our service model it is assumed that streams are correlated in space and time, as all streams in one site capture different sensory data from the same site at the same time. However, we assume *inter-stream coding independence* as each of the streams can be independently transmitted and rendered without the need of the other streams.

Inter-stream coding independence allows our service tier to use a publish-subscriber model. In our publish subscriber model output devices are the publishers and input devices are the subscribers. This model leverages user interest specific to the current activity taking place in the system to efficiently utilize the limited system resources, as only the streams that subscribed are transmitted. For example, 3D teleconferencing might only require one or two stereo cameras but several microphones where as exergaming might require a larger array of cameras to be transmitted and might not require audio.

In addition to data dissemination, the Service Tier is responsible for stream processing. I/O devices in 3D TI Systems are heterogeneous in terms of available computational processing power, as many of them are either small-factor devices (i.e., sensors), like accelerometers and heart rate monitors, or are already overloaded with complex algorithms required for proper data acquisition (e.g., 3-D reconstruction on Kinect cameras requires a Core i5 or equivalent processor to produce a video stream of 20 fps). Hence, CDGs are also responsible for offloading, transcoding and processing of streams.

Physical Tier

The physical architecture of a 3D TI System is composed of multiple geographically distributed sites interconnected through the Internet2 network infrastructure. Each of these sites has various input and output devices such as cameras, displays and sensors connected to them. Each device is accessible through a device-specific Application Programming Interface (API) provided by the device driver. Our model assumes that 3D TI devices are streaming devices, that is each devices produces a sequence of data frames spaced by a time interval. This concept is more formally defined in Section 3.2.

Each device is connected to a Content Delivery Gateway (CDG) at the local site either through a bus interface (e.g., USB or Firewire) or through a network connection (e.g., Ethernet). Content Delivery Gateway at the local site aggregates and disseminates the Bundle of Streams from the input devices to the output devices through the CDGs at the local and remote sites where they are rendered at the output devices.

The 3D TI System physical architecture is a closed distributed system in which the activities determine the session coordination and the lifecycle of I/O devices. The device lifecycle can be described as follow:1. Devices are plugged in by the user based on activity needs. 2. Devices are registered and configured by the system. 3. Devices are accessed and controlled by various system and application entities in the 3D TI System. 4. Devices are dynamically suspended by the system or unplugged by the user. 5. 3D TI System releases the resources allocated to the devices.

3.2 Streaming Model

In StreamOS, streams are defined as sequences of frames spaced by a constant time interval referred to as *period*. Formally, we can define a Periodic Stream S as an ordered pair: $S = (\{f_k\}_{k=1}^{n}, P)$, where f_k represents the k^{th} frame in a sequence of frames and P represents the period of the Stream. These Streams are aggregated into a collection of highly correlated streams from the same Physical Space, referred to as *Bundle of Streams* [27]. Formally, we define a Bundle of Streams as a set $BS_{PS} = \{S_j^{PS} | \, \mathrm{cor}_{PS}(S_i, S_j) \geq t \ \forall i, j \in [0, |SN^{PS}|)\}$, where PS denotes the Physical Space, $\mathrm{cor}_{PS}(S_i, S_j)$ is a function $\mathrm{cor}_{PS} : SN^{PS} \times SN^{PS} \rightarrow [0, 1]$ that measures the level of correlation between streams S_i and S_j, t is a threshold parameter that defines the minimum accepted level of correlation between streams in the same Bundle, and SN^{PS} represents the set of all streams in the Physical Space.

Also, our streaming model assumes *stream differentiation*. That is, based on the activity some streams are more important than others. For example, in 3D teleconferencing, front cameras might be more important than side cameras. More formally, we define Stream Differentiation as an integer value obtained

from a function that ranks how important a stream is within a 3D TI System as follows: $Y^{SN^{PS}} : SN^{PS} \rightarrow [0, |SN^{PS}|)$, where SN^{PS} is the set of all streams in the 3D TI and $|SN^{PS}|$ denotes the cardinality of the set SN^{PS}.

Furthermore, each stream S is mapped into two streams, an Input Stream S^{In} (Instream) and an Output Stream S^{Out} (Outstream), more formally denoted as $S \rightarrow (S^{In}, S^{Out})$. Based on their source and destination these streams can be classified into 4 types:

1. *Local Input Stream* (i.e., Local Instream): Data Streams being disseminated from the input devices (e.g., cameras and microphones) in the local site to the local CDG.

2. *Remote Input Stream* (i.e., Remote Instream): Data Streams being disseminated from the remote CDG to the local CDG.

3. *Local Output Stream* (i.e., Local Outstream): Data Streams being disseminated from the local CDG to the output devices (e.g., speakers and displays) in the local site.

4. *Remote Output Stream* (i.e., Remote Outstream): Data Streams being disseminated from the local CDG to the remote CDG.

Figure 3.3 shows the classification of streams in a 3D TI with 2 sites.

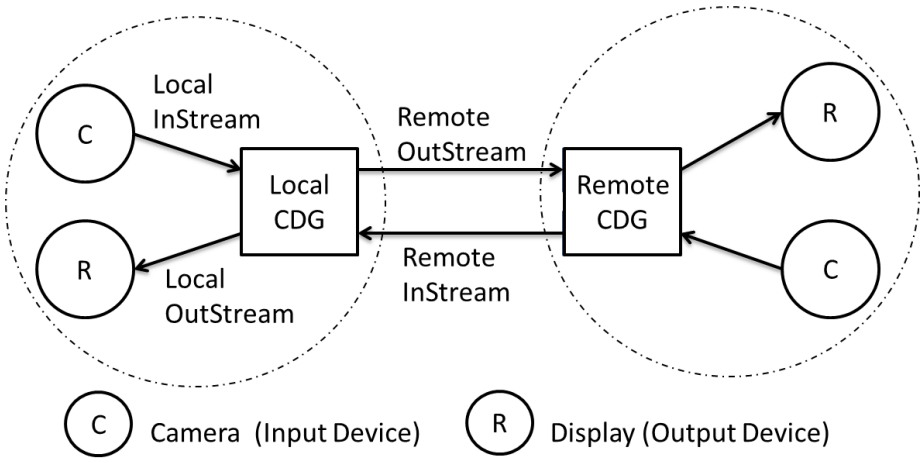


Figure 3.3: 3D TI Streaming Model

3.3 Activity Model

In StreamOS, activities are modeled as a cyber-physical concept with Physical and Computational elements. The Physical elements of the activity are 1. User Interaction, defined as the set of movements and gestures of the participants engaged in the 3D TI System and 2. Visual Space VS, defined as the most contributing color in the 3D TI scene. The Computational element of the activity

is the Stream Processing at the Content Delivery Gateway. Figure 3.4 shows the cyber-physical model of an activity in a 3D TI System with two sites.

The rationale behind associating Stream Processing at the Content Delivery Gateway as part of the activity, is that Stream Processing is highly dependent on the type of activity as different activities require different processing algorithms and functions. For example, Multiplayer Online Gaming might require authentication and object collision detection, while a Physiotherapy session might require encryption to ensure doctor-patient confidentiality.

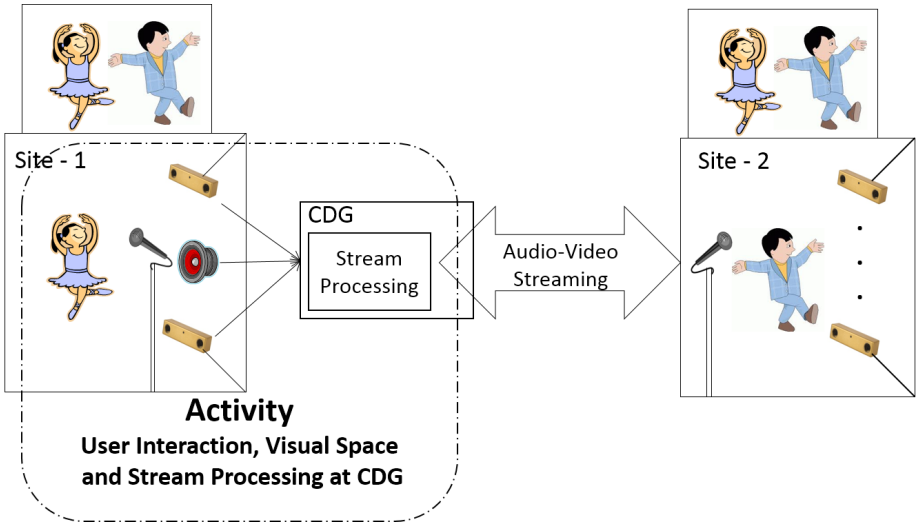


Figure 3.4: Cyberphysical Model of Activity

As part of the activity, each user specifies a *View of Interest*, composed by relevant streams the user is interested in viewing. For example, during a fencing session the user might prefer a lateral view over a frontal view of the virtual space. A View of Interest V can be more formally defined as a set of streams $V = \{S_i \mid S_i \in V \text{ if } o(S_i) = 1\}$, where $o(S_i)$ is a function $o : SN^{PS} \rightarrow \{0, 1\}$ that denotes if the user is interested in stream S_i and SN^{PS} denotes the set of all streams in the 3D TI.

3.4 Processing Model

Content Delivery Gateways are responsible for processing and disseminating Streams to the remote gateways. StreamOS models processing of Streams as Processing Functions within a Computing Pipeline. In our computing pipeline model each *Processing Function* is an element in a processing chain of frames arranged so that the input frame of a *Processing Function* is the output of the previous *Processing Function*. More formally, our computational pipeline can be defined as a composition monoid $(M, \circ)$ with a composition operation $F \circ G = F(G(x))$ in which F and G are Processing Functions defined as $F : \Phi_S \rightarrow \Phi_S$ and $G : \Phi_S \rightarrow \Phi_S$, where Φ_S is the super-set of all the

valid frames of a data stream S and the resulting streams of applying all the functions in the composition monoid. The set Φ_S is defined as follows: $\Phi_S = \{f_1, G(f_1), F(G(f_1)), f_2, \cdots, F(G(f_n))\}$. This algebraic structure allows us to seamlessly concatenate Processing Functions back to back.

Processing in 3D TI Systems can be categorized in 3 types of *Processing Functions*:

- **Frame Functions:** These functions operate over a single frame of a single stream. Example of this functions are Encryption and Compression.
- **Stream Functions:** These functions operate over the frames of a single stream. Example of this functions are QoS algorithms (e.g., Random Early Drop or Token Bucket).
- **Bundle Functions:** These functions operate over the frames of each of the streams forming a group or bundle. One example of this function is the Multi-Stream Synchronization [93].

Chapter 4

StreamOS Framework

As mentioned in Chapter 1 resource management in 3D TI Systems is a challenging task due to the Distributed Nature of 3D TI Systems, in which multiple geographical sites host a large scale of I/O Devices that produce Time and Space Correlated Streams. Furthermore, this Time and Space Correlated Streams are bound by tight QoS requirements in terms of skew, jitter and delay due the fast-paced Interactive Activities in 3D TI Systems. However, activities in 3D TI Systems are very diverse and each has a different set of QoS requirements that must be satisfied. In addition to this challenge, each activity also uses different devices with an Heterogeneous Non-Standard Interface.

To address these challenges, 3D TI Systems are usually implemented as an application running on top of a general purpose operating system like Linux or Microsoft Windows. Unfortunately, as this approaches do not provide complete support to address this challenges, the applications implementing the 3DTI system are cumbersome and difficult to maintain and deploy. The main obstacle on eliminating the burden of implementing resource and device management as user-applications in 3D TI systems is that current approaches to resource and device management do not consider all the challenges in 3D TI.

In this Chapter we present an overview of StreamOS, a Distributed Operating System that addresses the resource and device management challenges in 3D TI Systems through a holistic approach.

4.1 StreamOS Architecture

Stream OS is a Distributed Operating System for time and space correlated Streams in 3D TI Systems. It uses a holistic approach to resource and device management driven by the activities performed by the participants in 3D Teleimmersive Systems. In this holistic approach the activities, the users and the inherent characteristics of the 3D TI Systems impose dependencies in the resource and device management that must be considered across all the components of StreamOS at each different level of the functional model of the 3D TI:

- At the device level, the resource management is driven by the activity as different activities determine which devices will be used.

- At the stream level, as different activities, devices and topologies of the sites, create processing pipelines and time dependencies across streams and Bundle of Streams.
- At the task level, as different streams are mapped into groups of dependent and concurrent tasks with QoS parameters in terms of Computation Time and Period.
- At the job level, as frames from correlated streams must be scheduled concurrently to minimize skew across this streams.

We believe that this holistic approach is central to provide resource and device management in 3D TI Systems, as the characteristics of 3D TI Systems make it is necessary to provide a coordinated set of policies and mechanisms shared across all the components in the 3D TI System. The holistic approach is reflected in the architecture of StreamOS, residing across all the Content Delivery Gateways (Content Delivery Gateway) of the 3D Teleimmersive System. The Distributed Architecture significantly reduces several of the challenges in 3D TI as it allows to realize the holistic approach by providing mechanisms and policies to control and mange resources and devices across all the geographically distributed sites in the 3D TI System. Figure 4.1 shows the Distributed Architecture of StreamOS.

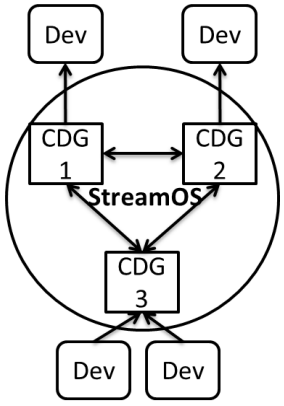


Figure 4.1: StreamOS Distributed Architecture

To address the resource and device management challenges described in Chapter 1, StreamOS follows a layered Component Architecture in which each of the subsystems of StreamOS solves these dependencies and challenges at different levels of the functional model. Figure 4.2 shows the Component Architecture of StreamOS. It is composed of 6 layered subsystems:

1. Cyberphysical Activity Layer: This layer represents the activity performed by the users in the 3D TI System. This layer is a cyberphysical layer. It includes the physical activity performed by the participant in the 3D TI System along with specific code necessary to manage the activity. For example, a virtual fencing exer-game can include code in this layer to allow

for immersive background scenarios and score keeping. This layer remains outside of StreamOS and represents a separate context space intended for end-user code.

2. Kratos - Activity Management and Detection Layer: This layer expands across all the Content Delivery Gateways in the 3D TI System. It is responsible for performing online detection of the 3D TI activity in the system based on metadata information. This layer provides an initial basis to address some the challenges that arise due to the Diversity of Interactive Activities by providing activity information to all the other layers of StreamOS. This layer is further described in Chapter 8.

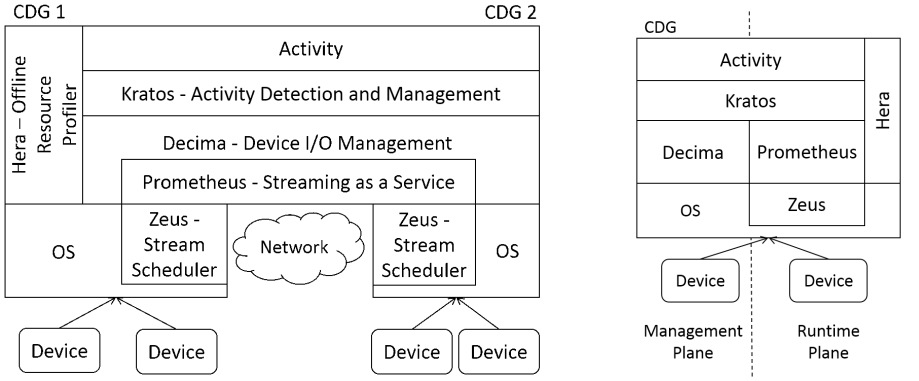
3. Decima - Device I/O Management Layer: This layer expands across all the Content Delivery Gateways in the 3D TI System. This distributed layer is responsible for allowing dynamic, seamless and universal interface to a large-scale of heterogeneous distributed stream-based I/O devices. This layer addresses the device management challenges that arise from Non-Standard Heterogeneous I/O devices being plugged and unplugged due to the Diversity of Interactive Activities. It also addresses the rapid changes in hardware of these Non-Standard Heterogeneous I/O. This layer addresses the problem of providing contextual support in terms of location and identification for time and space correlated groups of interactive streams that arise from the Distributed Nature of I/O devices. This layer is further described in Chapter 7.

4. Prometheus - Streaming as a Service Layer: This layer expands across all the Content Delivery Gateways in the 3D TI System. It provides end-to-end data delivery for I/O streaming devices. It provides setup, processing and resource management for groups of time and space correlated streams. This layer addresses the challenges that arise from Distributed Correlated Streaming I/O devices by providing streaming to Bundle of Streams across geographically distributed TI sites. It solves the challenge of interfacing with Non-Standard Heterogeneous I/O devices by providing a universal interface that can be accessed by a large range of multimodal devices. Finally, this layer provides user-defined processing functions to Bundle of Streams to address the challenges in terms of activity-driven stream processing due to the Diversity of Interactive Activities and the time and space Correlated nature of Streaming I/O devices. This layer is further described in Chapter 6.

5. Zeus - Real-time Stream Scheduler Layer: This layer resides at the bottom of the StreamOS architecture as a subsystem inside the traditional Operating System (e.g., Linux or Windows) of the Content Delivery Gateway. This layer provides efficient CPU Quality of Service to groups of correlated interactive streams (i.e., Bundle of Streams). It provides reservation-

based CPU guarantees for each bundle based on its streams' particular QoS requirement. This layer addresses the dependencies at the CPU Task Scheduling level that arise from Correlated Streaming I/O devices and also from the variability in the demand of CPU resources that arise from the Diversity of Interactive Activities. This layer is further described in Chapter 5.

6. Hera - Activity Based QoS Estimator: This service resides on top of the traditional Operating System and encompasses the other 5 layers of StreamOS. It provides offline estimation of the QoS requirements in terms of bandwidth and CPU utilization of the 3D TI System based on the activity and the type of devices used during the session. It addresses the contingency and variability of QoS requirements caused by the Diversity of Interactive Activities. This layer is further described in Chapter 9.



(a) StreamOS Component Architecture

(b) StreamOS Functional Planes

Figure 4.2: StreamOS Overview

4.2 StreamOS Session Protocol

In this Section we present an overview of the high-level protocol between the components of StreamOS during a 3D TI session. Detailed protocols are described in the specific Chapters for each subsystem. The high-level protocol of StreamOS can be divided in 3 main phases: 1. *Bootstrap Phase*, 2. *Running Phase* and 3. *Teardown Phase* . Figure 4.3 shows the high-level protocol of StreamOS. Below we describe the steps involved in each phase:

1. **Bootstrap Phase:** This phase includes all the initial setup of the 3D TI session. It involves the following steps:

   (a) Device Registration: After the devices are plugged in, they are registered through Decima and assigned a unique identifier. Details about Device registration are described in Chapter 7.

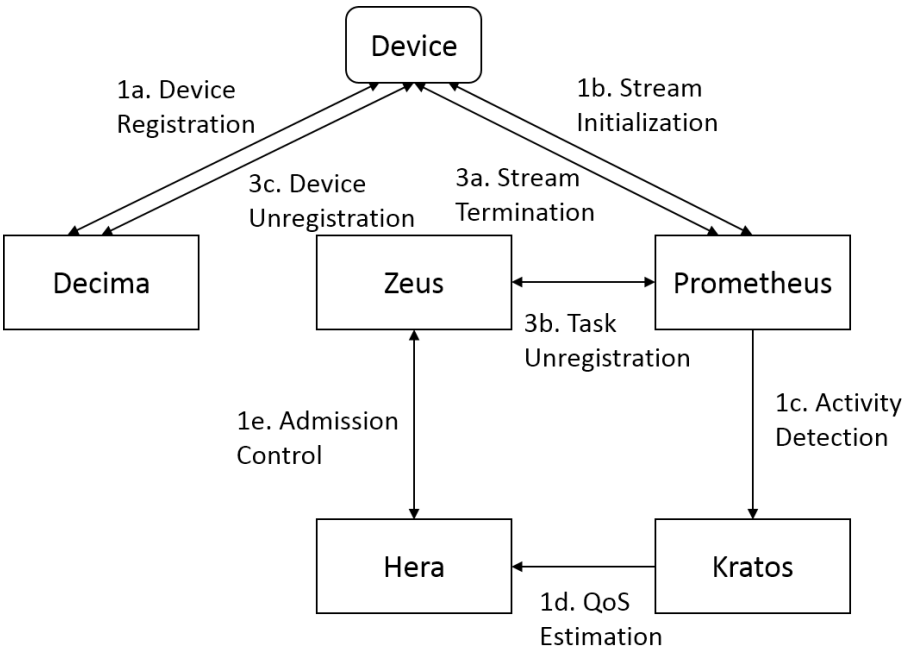


Figure 4.3: StreamOS High-Level Protocol

(b) Stream Initialization: Each device sends a Streaming request to Prometheus. At this time the device starts streaming and the stream is disseminated to remote sites. However, the system still cannot provide any Quality of Service guarantees (i.e., Best-Effort mode). Details about the Stream Initialization are described in Chapter 6.

(c) Activity Detection: During this step, participants enter the 3D TI scene and start interacting. Kratos monitors the state of the system and collects metadata related to 3D TI session to determine the activity engaged by the participants in the 3D TI System. Details about the Activity Detection are described in Chapter 8.

(d) QoS Estimation: After the activity is determined, the parameters about the 3D TI session (i.e., Session Description) are sent to Hera to estimate the QoS parameters (i.e., Session Parameters) of the 3D TI session. The obtained Session Parameters are used by Zeus to provide QoS guarantees or they can be used for offline resource estimation by system analysts and developers. Even that the 3D TI System is streaming during the QoS Estimation Hera uses previously stored data (i.e., System Profile) and therefore it is considered an offline profiling solution. Details about the QoS Estimation are specified in Chapter 9.

(e) Admission Control: The Session Parameters are sent to the Zeus Real-Time scheduler and Admission Control is performed on the Tasks associated with each group of Streams. If there are enough

resources then the Tasks are admitted. Admission Control and Real-Time Scheduling is described in Chapter 5. At this point StreamOS starts providing QoS guarantees (i.e., Real-Time mode).

2. **Running Phase:** This phase includes the interactions of the participants in the 3D TI session. It involves streaming of data from the input to the output devices.

3. **Teardown Phase:** This phase involves the teardown of the 3D TI session. It involves the following steps:

   (a) Stream Termination: Each device sends a Teardown request to to Prometheus. At this time the device stops streaming.

   (b) Task Unregistration: Tasks are unregistered from the Zeus Real-Time scheduler and are demoted to Best-Effort mode.

   (c) Device Unregistration: Devices are unregistered from Decima and the assigned unique identifiers are released. At this point the devices are unplugged.

Chapter 5

Zeus: Correlated Multi-Stream Scheduler

As mentioned in Chapter 1, a 3D TI System is composed of multiple capturing devices (e.g., camera arrays), rendering devices and Content Delivery Gateways (CDGs). Each one of these gateways is responsible of processing, aggregating and disseminating multiple codependent and correlated streams, called Bundle of Streams [27], from the capturing devices in the 3D TI. The real-time nature of data streams imposes tight Quality of Service (QoS) requirements in terms of delay, bandwidth and jitter that require the use of specialized soft real-time support.

Moreover, the multi-stream and multi-source nature of 3D TI requires extended synchronization and concurrent scheduling of each stream produced at capturing devices to minimize the skew between streams. For example, in a 3D TI, multiple cameras and sensors capture video, audio and sensory data from the local site, creating a Bundle of Streams. The Bundle of Streams is then distributed across multiple sites. Each of the streams in the bundle is correlated in time and must be synchronized and delivered with minimal skew between each other (*inter-stream synchronization*). Furthermost, at each CDG, a stream is mapped into two separate underlying streams (Instream and Outstream). Each stream in the bundle must be received, processed and disseminated, synchronously (*instream-outstream synchronization*). Figure 5.1 shows these stream synchronization requirements at the Content Delivery Gateway.

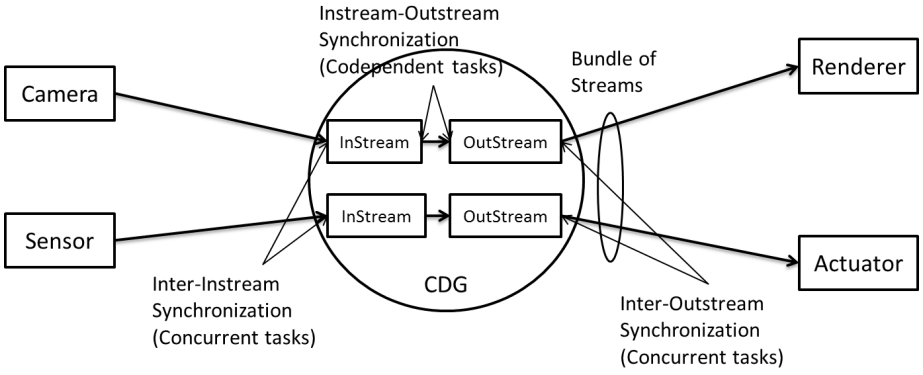


Figure 5.1: Synchronization requirements of the CDG in 3D TI

Gang scheduling [40] and co-scheduling address the problem of scheduling

of multiple tasks with concurrent constraints. These approaches have been well studied for best-effort, throughput-oriented multiprocessing like those proposed by Ousterhout et al. [41], Feitelson et al. [42] and Frachtenberg et al. [43]. However, they are not suitable for periodic soft real-time environments with tight QoS requirements like those found in 3D Teleimmersion.

Moreover, approaches to support real-time guarantees in commodity hardware (e.g., Rialto [22], DSRT [23], RTOS [24], RTLinux [44]). However, these approaches do not consider the dependencies of multiple time and space correlated streams and their translation into multiple time and space correlated processes as it is required by 3D TI Systems. These dependencies impose additional temporal constraints in terms of causality and concurrency to the traditional real-time scheduling in multiprocessor environments.

Gang EDF [50] considers concurrent constraints for real-time systems. In this work, Kato et al. propose a global EDF algorithm for multiprocessors in which they apply EDF policy to gang scheduling schemes. However, their algorithms are designed for sporadic parallel tasks and not for periodic tasks. Devices in 3D TI Systems are Streaming devices and therefore, sporadic approaches are unsuitable to provide QoS guarantees in 3D TI Systems.

Zeus is a soft real-time CPU scheduling architecture that resides in the Content Delivery Gateways (CDGs) of 3D TI Systems. Zeus supports Bundle of Streams with time dependency and concurrency constraints. In Zeus, each stream is mapped to a separate process of the traditional Operating System (e.g., Linux) and then these concurrent and dependent tasks are scheduled by the *Zeus Stream Scheduler*. Zeus provides efficient reservation-based CPU guarantees for groups of concurrent and dependent tasks running on multi-core architectures and based on its streams' particular QoS requirement. Hence, Zeus needs to support the following abstractions: 1. *Task dependency* to support instream-outstream synchronization, 2. *Task concurrency* to support inter-instream synchronization and inter-outstream synchronization.

As part of our solution to CPU scheduling of time and space correlated streams, we provide:

1. A novel process calculus that simplifies the specification and analysis of dependencies and concurrencies in time and space correlated process and stream systems.

2. Novel algorithms that provide scheduling for concurrent and codependent streams based on multi-core EDF policy.

3. Online algorithms to bound the skew between concurrent tasks based on the actual laxity of the task as computed at run-time.

5.1 Codependencies and Concurrency Constraints

In this section we analyze some of the challenges and requirements of our architecture:

Instream-Outstream Synchronization: Streams at Content Delivery Gateways must be received (Instream), processed and disseminated (Outstream), synchronously. This creates a dependency relationship across each of these stages and subsequently across each of the stream processes implementing them. The Zeus architecture needs to handle these chains of codependent stream processes and schedule them in the correct order.

Inter-Instream Synchronization: 3D TI Systems are composed of multiple devices that simultaneously produce streams of data with a temporal correlation. Streams from these devices must be received synchronously to minimize the skew across streams. For example, in a 3D TI System, two cameras that simultaneously capture video at a rate of 30 fps, each of the data units (e.g., video frames) of both streams at any instant in time are correlated and must be received synchronously. This creates a concurrent relationship between each of the data units of the two streams. This concurrent relationship translates to each of the receiving processes. The Zeus architecture needs to handle these stream processes with concurrent relationships and schedule them as close in time as possible to minimize the skew across these tasks.

Inter-Outstream Synchronization: Streams from multiple devices at the sending processes, must be sent synchronously to minimize the skew across these streams. The Zeus architecture needs to handle this concurrent relationship and schedule this process as close in time as possible. It is important to note that for our analysis Inter-Instream and Inter-Outstream synchronization can be generalized as Inter-Stream synchronization.

Real-Time Constraints: 3D TI Systems have tight requirements in terms of jitter and delay. These streaming systems are usually modeled as periodic processes with deadlines in which each data unit has certain CPU bandwidth requirements and must be processed before the deadline.

Next section describes more in detail the process model used by the Zeus architecture and also describes in detail the theoretical dependency and concurrency model.

5.2 Zeus Process and Group Model

In order to map a Stream into a Process in the traditional Operating System (e.g., Linux), Zeus defines the concept of a Task. As mentioned in Chapter 3, Content Delivery Gateways map each stream into two streams, an Input

Stream S^{In} (Instream) and an Output Stream S^{Out} (Outstream), therefore each Instream and each Outstream are mapped to different Tasks T_i and T_j in Zeus. Figure 5.2 shows an example of a Task mapping for a 3D TI System with 2 Instreams, each of them mapped to Task 1 and Task 3 respectively, and 2 Outstreams, mapped to Task 2 and Task 4.

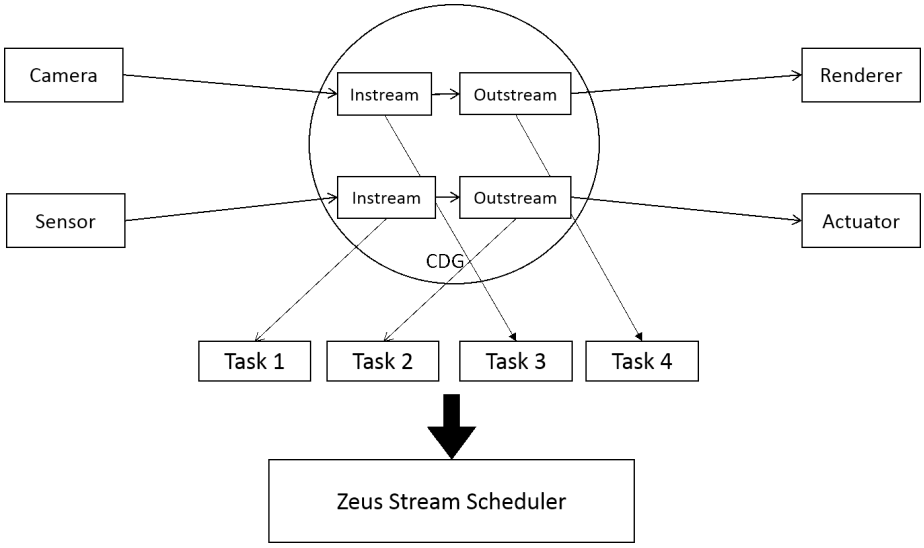


Figure 5.2: Stream to Task Mapping in Zeus

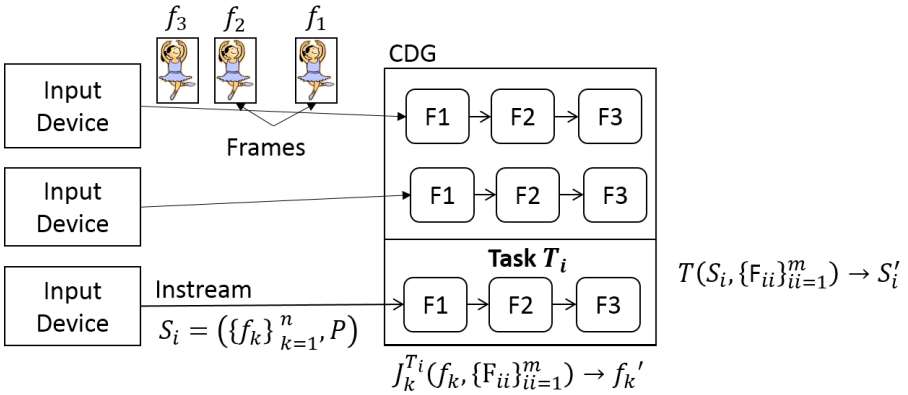
A Task is composed of a sequence of functions that are applied to a particular Stream. More formally, we define a Task T as a function $T(S, \{F_{ii}\}_{ii=1}^{m}) \rightarrow S'$, where S is a stream (i.e., Instream or Outstream), $\{F_{ii}\}_{ii=1}^{m}$ is a sequence of functions applied over the stream S and S' is the resulting stream. Each Task in Zeus is mapped to a different process in the traditional Operating System and therefore we will use the terms interchangeably.

To model the streaming nature of a Task, we will use a periodic process model, in which a process T_i running in the system with identifier i has a period P_i. In our process model, each frame f_k of a stream is associated with a Job $J_k^{T_i}$ that must be completed by Task T_i before a deadline D_i relative to the period of the process. Our model allows processes with deadlines smaller than the periods ($D_i < P_i$). Also, the process must specify the CPU bandwidth reservation C_i in CPU-time units required by the process to execute the Job J^{T_i} each period. Additionally, we define the release time E_i^k of a job as the earliest time that the current Job $J_k^{T_i}$ can be scheduled. In our periodic model, this time is defined as $E_i^k = P_i \times k$. Figure 5.3 shows the periodic process model of Task T_i over an Instream S_i

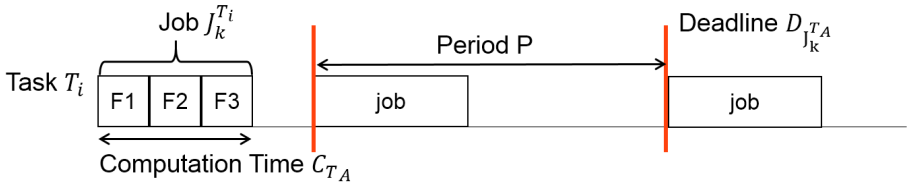
This model serves as the basis for specifying the QoS parameters in Zeus by a 5-tuple as follows:

$$T_i = (i,\ P_i,\ D_i,\ C_i, Y_i) \tag{5.1}$$

where i is the process identifier used by the OS and Y_i is the priority of the



(a) Concept of Periodic Process Model of Task T_i



(b) Timeline of Periodic Task T_i

Figure 5.3: Periodic Process Model in Zeus

process within the group. The priority Y_i of the process T_i is determined by the Stream Differentiation function as follows: $Y_i = Y^{SN^{PS}}(S)$, where $Y^{SN^{PS}}$ is the Stream Differentiation function rating the importance of a Stream S and S is the Stream associated with task T_i.

Due to the correlated nature of the streams (bundle concept) in 3D TI, each process in Zeus joins a group of processes G. Each Bundle of Streams is mapped to a group G. Each group G is composed of multiple processes with dependencies and concurrencies between them, as described in Section 5.1. More formally, a group of processes G is defined as a set: $G = \{T_i | T_i \in ST \wedge T_i \in G\}$, where T_i is a stream process and ST is the set of all processes in the system.

As part of our model, we introduce a process calculus to describe the dependencies and concurrencies within Zeus. Our process calculus should provide and advancement to the development of time correlated scheduling as it provides a simple mathematical basis to the specification and analysis of dependencies and concurrencies and should further aid in the modeling and design of scheduling systems with such constraints.

Our process calculus is composed of two binary relations: dependency (Instream-Outstream synchronization) and concurrency (Inter-stream synchronization). More formally, we define a dependency relation as

$$\begin{aligned} T_i \rightarrow T_j \doteq & \{(T_i, T_j) | (T_i, T_j) \in ST \times ST \wedge T_i, T_j \in G \\ & \wedge D_j \geq D_i + C_j \wedge FF_i^k <= E_j^k \wedge T_j \text{ depends on } T_i\} \end{aligned} \tag{5.2}$$

where E_j^k is the release time of job k for process T_j and FF_i^k is the finish time of job k for process T_i.

Our process model assumes that processes do not have any circular dependencies across them. Therefore, we restrict the set of processes to be partially ordered sets (posets), with many-to-many relationships. A process T_i can depend on many processes and also many processes can depend on T_i. From these assumptions it follows that:

1. The dependency relation is irreflexive:

$$T_i \nrightarrow T_i, \forall (T_i, T_i) \in ST \times ST \tag{5.3}$$

2. The dependency relation is antisymmetric:

$$T_i \rightarrow T_j \Rightarrow T_j \nrightarrow T_i, \forall T_i, T_j \in ST \tag{5.4}$$

3. The dependency relation is transitive.

$$T_i \rightarrow T_j \wedge T_j \rightarrow T_l \Rightarrow T_i \rightarrow T_l, \forall T_i, T_j, T_l \in ST \tag{5.5}$$

The previous properties can be proved trivially with set theory.

We consider these dependencies as *hard* dependencies in which if a process $T_i \rightarrow T_j$, then $P_i = P_j$ and if T_i does not complete before the next period then T_j does not need to be scheduled for that period. Zeus follows the Effective Deadline Assignment model [94] in which if $T_i \rightarrow T_j$ then $D_j \geq D_i + C_j$.

Additionally, we define a concurrency relation as follows:

$$\begin{aligned} T_i \mid T_j \doteq \{(T_i, T_j) | (T_i, T_j) \in ST \times ST \wedge T_i, T_j \in G \wedge \\ D_i = D_j + \epsilon \wedge E_i^k = E_j^k \wedge T_i \text{ is concurrent with } T_i\} \end{aligned} \tag{5.6}$$

where ϵ is the skew between the two processes.

From these assumptions it follows that:

1. The concurrency relation is reflexive:

$$T_i \mid T_i, \forall (T_i, T_i) \in ST \times ST \tag{5.7}$$

2. The concurrency relation is symmetric:

$$T_i \mid T_j \Rightarrow T_j \mid T_i, \forall T_i, T_j \in ST \tag{5.8}$$

3. The concurrency relation is transitive.

$$T_i \mid T_j \wedge T_j \mid T_l \Rightarrow T_i \mid T_l, \forall T_i, T_j, T_l \in ST \tag{5.9}$$

Similarly, these properties can be proved trivially with set theory.

Concurrent constraints on the other hand are considered as *soft* constraints in which if processes $T_i \mid T_j$ then they will be scheduled concurrently but not necessarily simultaneously. This property is important as it allows our scheduler to use the residual budget to define a concurrency budget based on the *laxity of a task*. Therefore, Zeus will schedule the processes as close as possible in time as long as neither of them causes either process or other processes in the system to miss a deadline. Also, we assume that if $T_i \mid T_j$ then there exists a synchronization point at $P_i = P_j$ and $D_i = D_j + \epsilon$. We allow each process to be concurrent with any number of processes.

5.3 Zeus Architecture

In order to implement the Stream Process and Group Models described above and to successfully address the time dependency and concurrency constraints described in Section 5.1, our Zeus architecture consists of 5 main entities: a Constraints Verifier, Admission Control, Partition Manager, Constraint Coordinator and Soft Real-Time Scheduler.

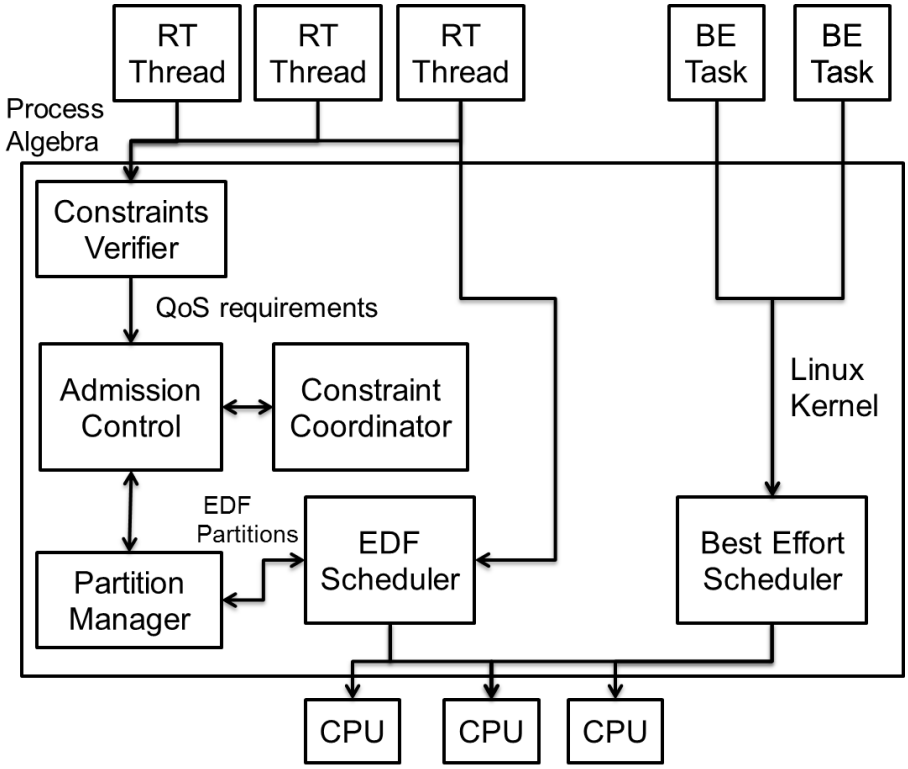


Figure 5.4: Architecture of Zeus and its 5 main entities

Figure 5.4 shows the architecture of Zeus running soft Real-time processes and also running Best-Effort processes. Zeus is divided into 3 phases based on the life cycle of a real-time process:

1. **Registration Phase:** The registration phase consists of the following steps:

   (a) Each real-time process T_i of the group G specifies its QoS parameters, dependencies and concurrent constraints to Zeus.

   (b) Dependencies and concurrency constraints of the group G are verified by the Constraints Verifier.

   (c) Process group G gets admitted by the Admission Control

   (d) Zeus Partition Manager assigns each admitted process T_i of the group G to a CPU κ.

2. **Running Phase:** Each processes T_i in the group G execute the soft real-time code under the policies of the Soft Real-Time Scheduler according to the specified constraints enforced by the Constraint Coordinator.

3. **Shutdown Phase:** The group G unregisters each task T_i and exits.

The rest of this section describes each component and its algorithms.

5.3.1 Constraints Verifier

The Constraints Verifier is responsible for collecting the concurrency and dependency constraints of each Group of processes G in the system. Constraints are specified using the process calculus described in Section 5.2.

The Constraints Verifier analyzes these dependencies and concurrencies to ensure that they are valid and that they follow the described stream process model in terms of periods and deadlines. During the *Registration Phase*, the Constraint Verifier builds a dependency and concurrency graph of the group G joining the system, referred to as the Constraint Graph (CG). In this graph the processes are represented by the vertices, the dependencies are represented as directed edges and the concurrency constraints are represented as undirected edges. If process T_j depends on process T_i, then the Constraint Graph will have a directed edge from vertex T_i to vertex T_j. Processes without dependencies or concurrencies appear in the graph as disconnected vertices. A similar approach to dependencies and concurrencies has been proposed in the data mining domain by Dahlhaus et al. [95]. More formally, we define the Constraint Graph for group G as follows:

$$\begin{aligned} &\mathrm{CG}(G) = (V, \overline{E}, \overrightarrow{E}) \\ &V = \{T_i | T_i \in G\} \\ &\overline{E} = \{(T_i, T_j) | T_i \mid T_j\} \\ &\overrightarrow{E} = \{(T_i, T_j) | T_i \rightarrow T_j\} \end{aligned} \tag{5.10}$$

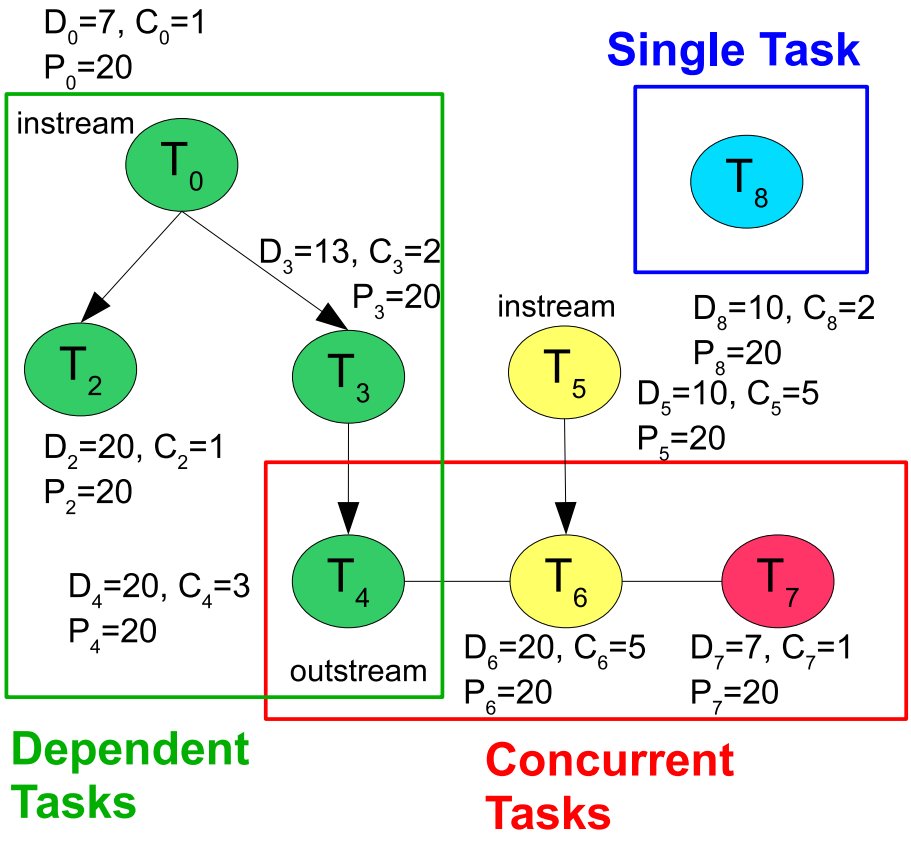


Figure 5.5: Constraint Graph showing single processes and processes with dependency and concurrency constraints

Additionally, the Constraint Verifier maintains a Constraint Graph of the current process set ST in the system. This graph is defined as:

$$\mathrm{CG}(S) = \bigcup_{G \subset ST} \mathrm{CG}(G) \tag{5.11}$$

Figure 5.5 shows a process set where processes T_3 and T_2 depend on process T_0, and T_4 depends on process T_3. Also, the Figure 5.5 shows processes T_4, T_6 and T_7 share a concurrency constraint and process T_8 is a single process with no dependencies or constraints.

During the *Registration Phase*, the constraint verifier traverses the Constraint Graph (CG) of the joining group G using Algorithm 1 to ensure that the specified dependencies are feasible. For each dependency $T_i \rightarrow T_j$ the algorithm verifies that $D_j >= D_i + C_j$ and that $P_i = P_j$, otherwise the group is rejected. Similarly the algorithm verifies for each concurrency $T_i \mid T_j$ that $P_i = P_j$, otherwise the group is rejected.This verification is independent on the CPU assignment or the feasibility of meeting the deadlines.

For example, in the group represented by the Constraint Graph shown in Figure 5.5, the Constraint Verifier needs to check that for process T_4: $D_4 >=$

Algorithm 1 Algorithm to verify the constraints of group G

Input: $\mathrm{CG}(G) = (V, \overline{E}, \overrightarrow{E})$
Output: 'admit' or 'reject'
  for all $(T_i, T_j) \in \overrightarrow{E}$ **do**
    if $D_j < D_i + C_j$ **then**
      return 'reject'
    end if
    if $P_i \neq P_j$ **then**
      return 'reject'
    end if
  end for
  for all $(T_i, T_j) \in \overline{E}$ **do**
    if $P_i \neq P_j$ **then**
      return 'reject'
    end if
  end for
  return 'admit'

$D_3 + C_4$, and $P_4 = P_3$. This check is repeated for each dependency. For the case of concurrencies, in this example, it must check that $P_4 = P_6 = P_7$

5.3.2 Partition Manager

The Partition Manager is responsible for assigning a CPU to each of the processes T_i in the new group G. The Partition Manager uses heuristic described in Algorithm 2 to assign a CPU for the new process. This heuristic is based on the Worst Fit algorithm [96] for bin-packing. Our heuristic can be described as follows: 1. If the new process T_i depends on process T_j ($T_j \rightarrow T_i$) and T_j is running on CPU κ, the system assigns process T_i to CPU κ. 2. If the new process T_i has a concurrent constraint with process T_j ($T_j \mid T_i$) then the Partition Manager uses the Worst Fit algorithm. 3. If the new process T_i is a single process then the Partition Manager uses the Worst Fit algorithm.

In the Worst Fit algorithm, we compare the utilization of each CPU and we pick the one with the lowest value. The Work Fit heuristic allows us to balance the load evenly across each of the CPUs in the system, which is a desirable property in modern CPU architectures with mid-level split caches. However, this heuristic does not itself guarantee that the system can meet the deadlines of the group G.

For example, in the group represented by the CG shown in Figure 5.5, the Partition manager would assign processes T_0, T_2, T_3, T_4 to the same partition and processes T_6 and T_7 to different partitions.

5.3.3 Admission Control

The Admission Control is responsible for deciding whether the group should be admitted as part of the process set of the system or not and whether the

Algorithm 2 Algorithm to assign a CPU partition to each process $T_i \in G$

Input: $\text{CG}(G) = (V, \overline{E}, \overrightarrow{E})$, a vector $(R_0, \cdots, R_K)$ containing the residual budget available for each CPU
Output: a vector $(\kappa_0, \cdots, \kappa_{\|V\|})$ containing a CPU assignment κ_i for each process T_i in G

```
for all T_i ∈ V do
  if ∃(T_j, T_i) ∈ E→ then
    κ_i ← κ_j {Assign T_i to T_j's CPU}
  else
    if ∃(T_i, T_j) ∈ E¯ then
      κ_i ← κ_l, such that l ≠ j' ∀(T_i, T_j') ∈ E¯ and R_l = max_{l'=0}^{K}(R_l')
    else
      κ_i ← k_l, such that R_l = max_{l'=0}^{K}(R_l')
    end if
  end if
end for
return (κ_0, ··· , κ_{‖V‖})
```

specified dependency constraints can be satisfied by the scheduler. For each process in the group, the Admission control decides on these 3 possible outcomes: 1. The system cannot satisfy the QoS requirements required, in which case the process will not be admitted. 2. The QoS requirements are guaranteed but the dependency constraints cannot be satisfied, in which the process may decide to either withdraw itself from the system or continue the registration phase as a single process. 3. Both the QoS requirements and the dependency constraints are guaranteed.

In order to decide on these three outcomes, the Admission Control uses Algorithm 3. This algorithm takes the group of processes sorted by priority Y_i. Then it verifies in Breadth First Search (BFS) order that each process can be admitted as part of the group. To do that, each process must satisfy two constraints: 1. For all T_j, such that $T_j \rightarrow T_i$, then T_j must satisfy the single process admission control constraint. 2. The single admission control constraint for each process T_i must be satisfied. Algorithm 3 describes more formally the steps involved in the admission control of a group G.

For example, in the group represented by the CG shown in Figure 5.5, the group Admission Control would verify the processes in group G in BFS order: T_0, T_8, T_5, T_7, T_2, T_3, T_6 and T_4.

In order to verify that each process meets the admission control individually, Algorithm 3 uses the Single process Admission Control (e.g. Single_Admission()), formally described in Algorithm 4. In the next paragraph we briefly describe this algorithm:

Given a partition assignment for process T_i, the Admission Control temporarily inserts the processes of the group G in their corresponding assigned CPU partition and then uses the processor demand analysis on all the processes in each CPU k in the system:

Algorithm 3 Algorithm to decide the admission of a group G

Input: CG$(G) = (V, \overline{E}, \overrightarrow{E})$, a vector $(\kappa_0, \cdots, \kappa_{\|V\|})$ containing a CPU assignment κ_i for each process T_i in G
Output: a vector $(a_0, \cdots, a_{\|V\|})$ containing the admission decision for each process $T_i \in G$

```
for all T_i ∈ V ∧ a_i == 'undecided', sorted by directed BFS over E→ and priority Y_i do
  if ∃(T_j, T_i) ∈ E→ then
    if a_j == 'admit' then
      if Single_Admission(T_i) == 'admit' then
        a_i ← 'admit';
      else
        a_i ← 'reject';
      end if
    else
      if Single_Admission(T_i) == 'admit' then
        a_i ← 'single_admit';
      else
        a_i ← 'reject';
      end if
    end if
  else
    if Single_Admission(T_i) == 'admit' then
      a_i ← 'admit';
    else
      a_i ← 'reject';
    end if
  end if
end for
return (a_0, ··· , a_||V||)
```

$$\forall L \in \Delta, L \geq \sum_{T_j \in ST_\kappa} \left\lfloor \frac{L - D_j}{P_j} + 1 \right\rfloor \cdot C_j \tag{5.12}$$

where L is an instance in time and Δ is the set of all absolute deadlines in the hyper-period of the process set ST in the system. To calculate the hyper-period of all the processes in the system, denoted as HP, Zeus performs the following calculation

$$\text{HP} = \operatorname*{lcm}_{P_j \in G \cup ST}(P_j) \tag{5.13}$$

After admission of the group, the admission control is responsible for calculating the residual budget for each CPU (e.g., laxity), defined as the amount of CPU time units available in the CPU at any instant in time. The Admission Control performs the following computation of the residual budget for each CPU κ:

Algorithm 4 Algorithm to decide the admission of a single process T_i: Single_Admission()

Input: a process T_i, the process partition assigned ST_κ, the process group G and the process set of the system ST
Output: the admission outcome: 'admit' or 'reject'

```
HP = lcm_{P_j ∈ G∪ST}(P_j)
Δ = The set of all absolute deadlines ∀T_j ∈ G ∪ S for 0 ≥ L ≥ HP
for all T_i ∈ ST_κ do
  if ∃L ∈ Δ, L < Σ_{T_j ∈ ST_κ} ⌊(L − D_j)/P_j + 1⌋ · C_j then
    return 'reject'
  end if
end for
return 'admit'
```

$$\forall L \in \Delta, R_\kappa^L = L - \sum_{T_j \in ST_\kappa} \left\lfloor \frac{L - D_j}{P_j} + 1 \right\rfloor \cdot C_j \tag{5.14}$$

The equation above represents the residual budget at each deadline $L \in \Delta$ for a partition ST_k. As a lower bound on the available residual budget in the system at any time in the hyperperiod HP we can take the minimum of such residual budgets such that

$$R_k = \min_{L \in \Delta}(R_\kappa^L) \tag{5.15}$$

Finally, the admission control is responsible of synchronizing the Release Time E_i of all the newly admitted processes T_i in the group G.

5.3.4 Soft Real-Time Scheduler

The Soft Real-Time Scheduler is a multi-core soft real-time scheduler based on the Earliest Deadline First policy. It is responsible for finding a feasible schedule given the current process set ST in the system and their QoS parameters. The Real-Time Scheduler is responsible only for scheduling real-time stream processes. Best-effort processes are scheduled by the best-effort scheduler implemented by the OS. Scheduling decisions of the Real-Time Scheduler will take precedence over the scheduling decisions of the best-effort scheduler. If the Real-Time Scheduler cannot find a pending Real-Time job then it will forfeit its decision and yield control to the Best-Effort Scheduler. To prevent starvation of Best-Effort processes, we allow the user to specify a fixed reservation for this processes.

The Real-Time Scheduler uses a *Partitioned EDF* [97] approach. Partitioned EDF requires the process set to be divided into K partitions, where K is the number of CPUs in the system. eus stripes processes with concurrencies across different CPUs and keeps processes with dependencies in the same CPU, as described in Algorithm 2. Then the scheduler applies the EDF policy to each

of them independently. Zeus uses a partitioned approach due to its simplicity, higher acceptance ratio of random processes and lower overhead [98].

5.3.5 Constraint Coordinator

The Constraint Coordinator is responsible of enforcing the dependencies and concurrencies specified by each group G. It is responsible of requesting rescheduling of a process if the dependencies have not been satisfied or if one of the processes in the concurrent group is not ready to be scheduled, (e.g., some of its dependencies have not been satisfied or its assigned CPU is scheduling other process). In the case of dependencies, for a process T_i running on CPU κ, the constraint coordinator will enforce the constraint until all the dependencies have completed. If a user-defined timeout expires, then the process will exit and the user will be notified of the error. However, for the case of concurrencies, it will do it for as long as no process in the process set ST_κ including T_i will miss its QoS guarantees by delaying process T_i.

5.4 Zeus Scheduler Design

In this section we discuss in detail the architecture of Zeus from a process life cycle perspective.

5.4.1 Registration Phase

During the *Registration Phase* of the group G, each process is responsible of specifying its QoS parameters defined in Section 5.3. Also, the group specifies its concurrency and dependency constraints to the Constraint Verifier. After the Constraint Verifier checks for the feasibility of these constraints, it sends the CG graph to the Admission control to perform group admission and if this fails, perform single admission of the failing processes in the group.

Finally, the Release Time E_i^0 of all the processes T_i in the group G are synchronized and they enter the Execution Phase.

5.4.2 Execution Phase

During the Execution Phase, the scheduler is responsible for enforcing the EDF policy for each process partition in the system. During the execution phase each of the processes can be in 1 of 6 states. Figure 5.6 shows a Finite State Machine of the possible 6 states for a process: 1. *Initial* state in which the process waits for the rest of the processes in the group to join. 2. *Ready* state which indicates that a process is ready to run for the current period. 3. *Wait_Deps* state in which a process ready to run for the current period waits for the processes it depends on. 4. *Wait_Con* state in which a process ready to run for the current period waits for the other concurrent processes to reach this state. 5. *Run* state

in which a process is currently being executed 6. *Sleep* state in which a process sleeps until the next period.

Initially, after admission of the group G, the process is put in the *Initial* state. In this state processes wait until all the dependencies and concurrencies reach this state. If a user-defined timeout expires then the processes exit and the user is notified of this error. This allows for a synchronized start of the constrained processes. When all the constrained processes reach this state, they are simultaneously transitioned to the *Ready* state.

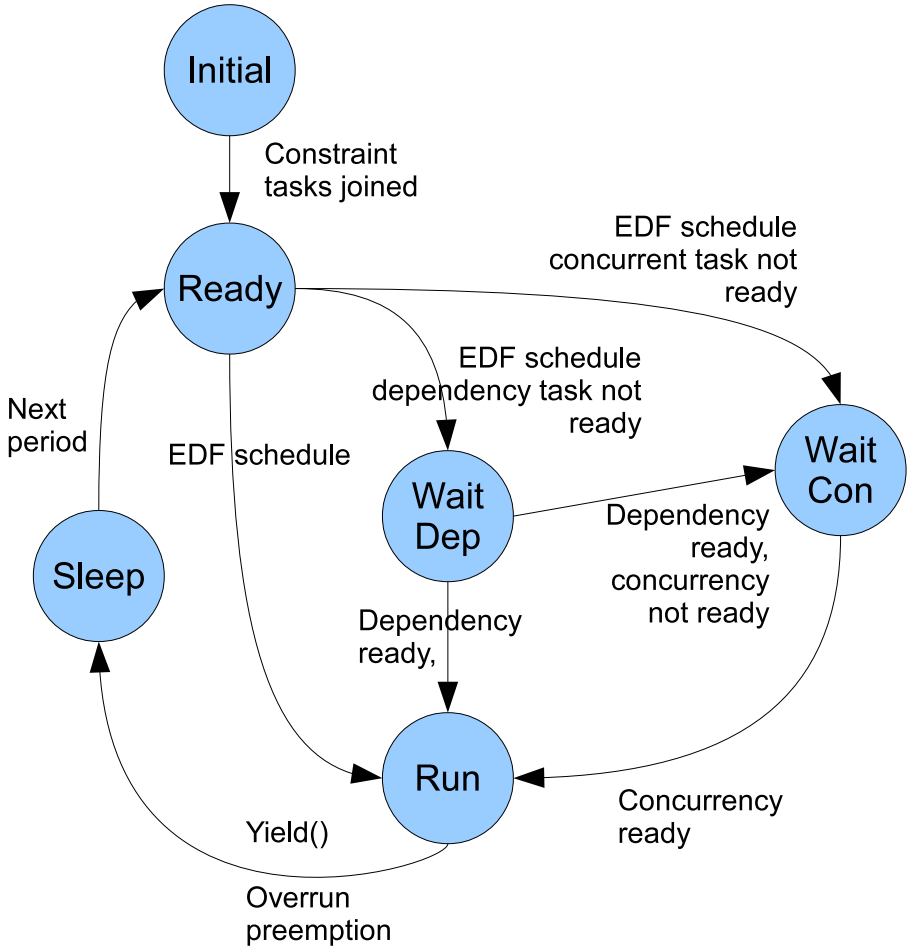


Figure 5.6: Finite State Machine for the state of a process in Zeus

When a process reaches the *Ready* state, it is inserted into the scheduler Ready queue and it is available for scheduling under the EDF policy. When a process in the *Ready* state is picked by the scheduler, the Constraint Coordinator verifies that all the dependencies have completed the job for the current period. For example, if process T_i is picked by the scheduler and $T_j \rightarrow T_i$ and $T_l \rightarrow T_i$, the Constraint Verifier will check that both T_j and T_l have completed the job for their current period. If any of the constraints has not completed its job then the scheduler will move process T_i to the *Wait_Dep* state and be removed

from the Ready queue. In the *Wait_Dep* state a process will sleep until all the dependencies are satisfied. As soon as the dependencies are satisfied the process is reinserted in the Ready queue and is available for scheduling by the EDF scheduler.

If a process T_i picked by the scheduler has no dependencies or all the dependencies have completed their current period, then the Constraint Coordinator verifies that all the concurrency constraints have reached the *Wait_Con* state. If one of the concurrency constraints has not reached this state, then process T_i is transitioned to the *Wait_Con* state and it is removed from the Ready queue. However, if all the concurrency constraints have reached the *Wait_Con* state, then the Constraint Coordinator inserts all the concurrency constraints back into the Ready queue.

As mentioned in Section 5.2, the Zeus architecture considers all the concurrency constraints as *soft* constraints. It does not perform admission control on them, which simplifies the complexity of the system. Instead, the Zeus architecture uses the residual budget from each CPU partition and distributes them evenly across all the processes with concurrency constraints in the partition (i.e., concurrency budget).

Zeus allocates the concurrency budget R_κ evenly across each job in partition κ with concurrent constraints in the hyperperiod HP. If the allocated concurrency budget depletes for the waiting job then the process is inserted in the Ready queue and it is available for schedule even if the other concurrency constraints have not reached the *Wait_Con* state. This policy simplifies the scheduling and the admission control and also allows the processes to preserve certain dynamism in their CPU usage. The *Wait_Con* state is implemented similarly as a *timed-barrier* in which n processes must reach the barrier before they are released, but each process has a waiting timeout. If the timeout expires for that waiting process then it is released without further waiting for the other processes.

If the process T_i has no dependency or concurrency constraints, the Constraint Coordinator will let the process run as soon as the EDF scheduler picks it from the queue. Once a process is allowed to run by the Constraint Coordinator, the state of the process is updated to the *Run* state in which it will run until the process completes its job for the current period. Processes that finished their job are required to notify the Zeus architecture through the yield() system call. At this time the process is removed from the Ready queue and its state is transitioned to the *Sleep* state.

If the process does not finish its job by the reserved CPU bandwidth C_i, the Overrun Timer in the EDF scheduler preempts the process to Best-Effort mode, in which it is allowed to run without interfering with the reserved allocation of the other Real-Time processes in the system. This involves removing the process from the ready queue. As soon as the misbehaving process finishes its job, the state of the process is updated to the *Sleep* state.

Processes in the *Sleep* state remain dormant until their next period. At the beginning of their period, their state is updated to the *Ready* state.

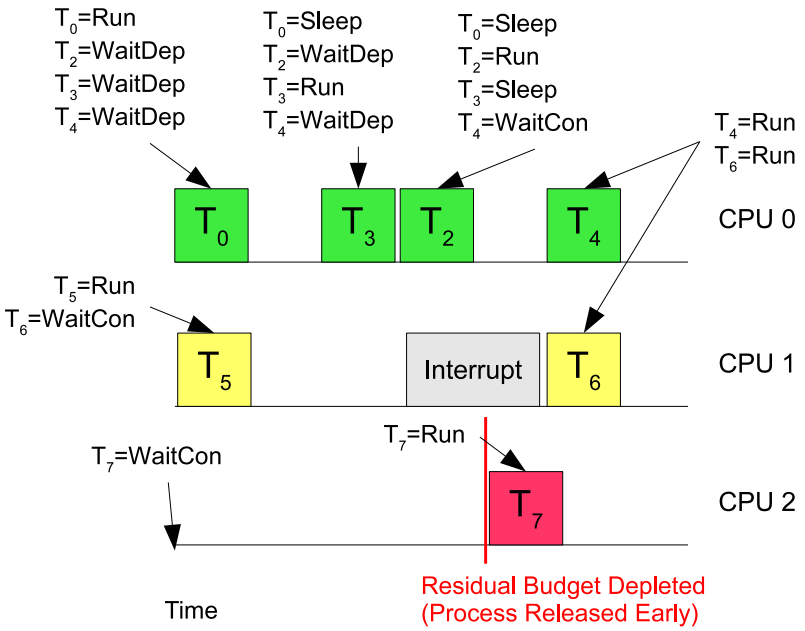


Figure 5.7: Example of the execution and process states

Figure 5.7 shows the execution and process states for the example process set used in Figure 5.5. Assuming that the partition $ST_0 = \{T_0, T_2, T_3, T_4\}$, then processes T_2 and T_3 will wait in the *Wait_Dep* state until process T_0 completes its job and process T_4 will wait in the *Wait_Dep* state until process T_3 finishes. Additionally, the processes T_4, T_7 and T_6 will wait in the *Wait_Con* state until the three processes reach this state. As mentioned above, dependency overruns, interrupts or kernel activity can cause delay in the schedule making it impossible to meet the inter-stream synchronization. As an example, Figure 5.7 shows the Residual Budget R_2 (i.e., concurrency budget) for CPU 2 depleting and causing process T_7 to be released earlier. This violates the inter-stream synchronization for tasks T_4, T_7 and T_6 but prevents other processes in CPU 2 from missing their deadlines.

5.4.3 Shutdown Phase

During the shutdown phase of process T_i, the process requests to be removed from Zeus. The Zeus scheduler recalculates the Residual Budget for the K CPU partitions and updates the Constraints Graph in the Constraints Verifier by removing any reference to process T_i. It also must release any waiting process in the *Wait_Dep* and *Wait_Con* states that are waiting only for process T_i.

A group shutdown is only a sequential shutdown of every single process in the group G.

5.5 Experimental Evaluation

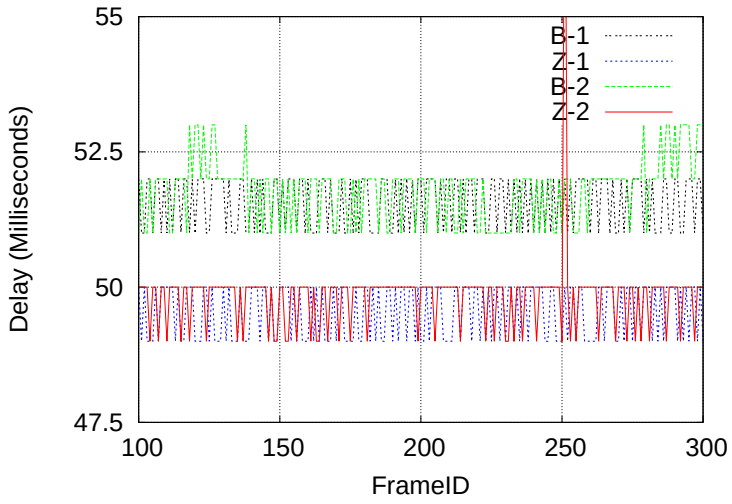
We evaluate the performance of Zeus in a real 3D Teleimmersion (3DTI) System. We validate Zeus in a Content Delivery Gateway (CDG). The CDG disseminates the incoming streams to a renderer device. The CDG is composed of an Instream process and an Outstream process for each stream. To make the experiments repeatable, we use a recorded creative dance performance stored in a computer running a vanilla Linux Kernel 2.6.30. This computer delivers the streams to the CDG. All the measurements are performed in the CDG. Machines used for all experiments were Dell Precision 690 with a Quad-core Intel Xeon Processor and 2 Gb of RAM. The frame rate of the recorded performance is 20 fps. All the experiments run for 15 minutes. However, due to space limitations on the graphs, we show only 200 frames, except where indicated.

For Experiment 1, we evaluate the total delay between frames as measured at the CDG. We compare the delay of the CDG running under Linux 2.6.30 and the delay of the CDG under Zeus. For this experiment we vary the number of cameras between 1 and 4. Figures 5.8(a) and 5.8(b) show the delay in milliseconds for each frame. We can observe that the delay incurred by the CDG under Zeus (Z-1, Z-2, Z-3, Z-4) is smaller in all cases.

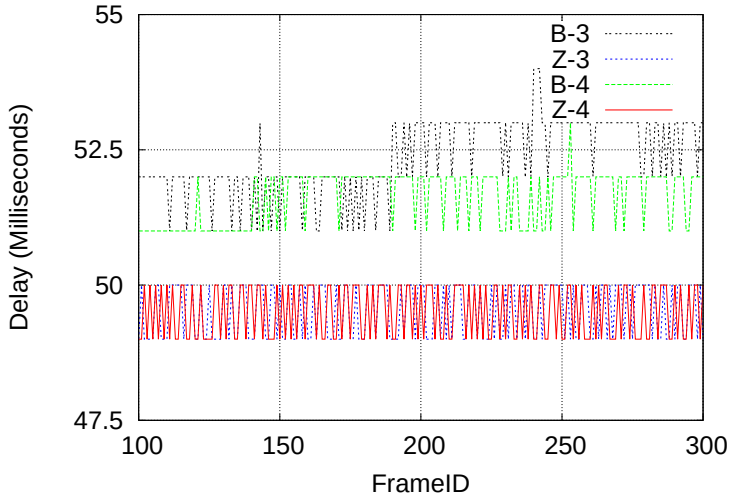
For Experiment 2, we evaluate the total delay between frames as measured at the CDG under best-effort overload. For the best-effort overload we use the *stress* program [99] with 16 threads running simultaneously. We vary the number of cameras between 1 and 3. We compare the results under Zeus and under Linux. Figure 5.9 shows the results for this experiment. We can observe that the delay incurred by the CDG under Zeus (Z-1, Z-2, Z-3) is smaller in all the cases. Also, we can observe that the performance of the Best-Effort (B-1,B-2,B-3) is severely effected when compared to the results from the Experiment 1. This shows that Zeus effectively can isolate Real-Time from Best-Effort workloads.

In Experiment 3, we evaluate the jitter measured at the CDG of one stream with and without best-effort overload. We compare the results under Zeus and under Linux. We show measurements for 300 frames. Figure 5.10 shows the results for this experiment. We can observe that the jitter for Zeus with (Z-Stress) and without stress (Z-No-Stress) is fairly constant near zero. The graph shows a few spikes for the Zeus scheduler possibly caused by interrupts, cache misses, page faults or kernel events. However, the results for Best-Effort show a much higher value for jitter.

Finally, for Experiment 4, we measure the delay at the Outstream process in the CDG. The Outstream process of the CDG involves reading the camera frame from the buffer and sending it to the network. The buffer is protected by a semaphore that is locked if the buffer is empty (Lock Delay). After sending the data, the Outstream process calls the yield() function (Scheduler Delay). We measure each of these delays along with the total delay. We measure these



(a) Delay of 1 and 2 simultaneous streams



(b) Delay of 3 and 4 simultaneous streams

Figure 5.8: Zeus Evaluation (Experiment 1)

delays for the Zeus scheduler (Z) and the Linux scheduler (B). We can observe in Figure 5.11 that the Scheduler Delay for the Zeus architecture (Z-Sched) is similar to the period of the video stream (50 ms). Also, we can observe that due to the dependency constraint registered to Zeus, the Lock Delay is close to zero. This yields very high efficiency because we avoid waiting on a lock and possibly context switching during this waiting time. The Outstream process running on Zeus sleeps in the *Wait_Dep* state until the data is ready in the buffer (e.g., the Instream process completes for the period). However, for the case of the Best-Effort, the scheduler delay is close to zero, but the Lock Delay is very high. This is caused by a bouncing effect in which the Outstream wakes up out of sync and checks for the data in the buffer. Because the buffer is empty, the process sleeps waiting for the lock. This bouncing effect contributes to an

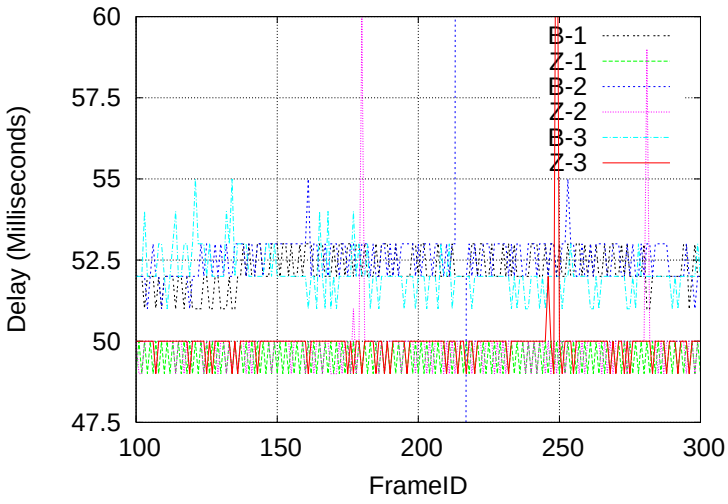


Figure 5.9: Delay of 1 to 3 streams with best-effort overload

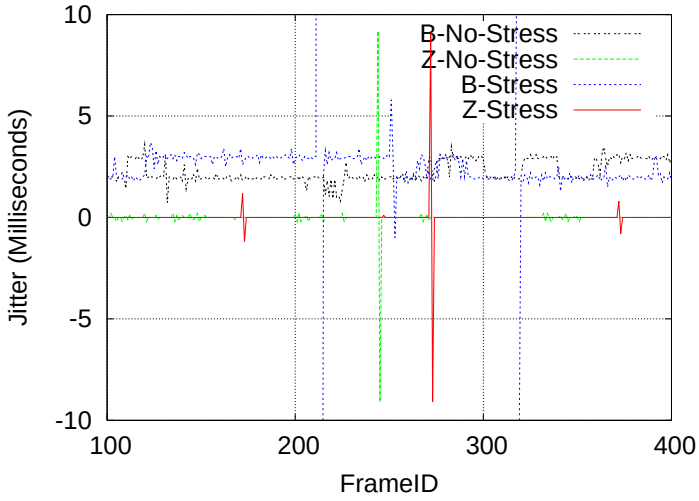


Figure 5.10: Jitter of a stream with and without best-effort overload

increased delay in the CDG under Linux, as shown by Experiment 1.

5.6 Conclusion

As part of our solution, we introduce a Process Calculus to model the relations of concurrency and dependency between time and space correlated streams in 3D TI Systems. This novel model provides powerful notation that allows modeling the complex constraints of groups of streams in 3D TI Systems. We believe that our model will contribute to the advancement in the area of real-time co-scheduling as future work can leverage this mathematical to further understand and model stream and task concurrencies and dependencies not only in 3D TI Systems but in correlated multi-stream systems.

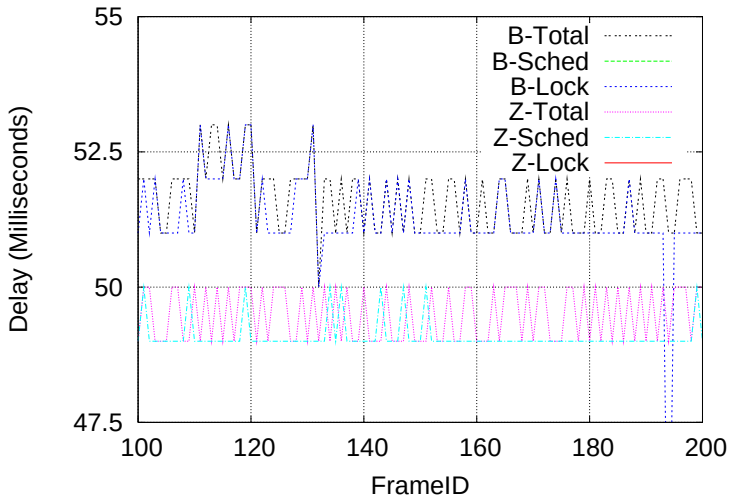


Figure 5.11: Delay Decomposition of a stream

Our solution also introduces Zeus, a soft real-time CPU scheduling architecture for 3D TI Content Delivery Gateways (CDGs) to support Bundle of Streams on multi-core architectures. In Zeus we use our Process Calculus as a basis to design a novel scheduling algorithm for concurrent and codependent tasks in multi-processor systems based on the partitioned Earliest Deadline First algorithm [26]. Our novel scheduling algorithm uses a concurrency budget based on the laxity of the task to minimize the amount of the skew between tasks depending on their actual running time. As part of our contribution we introduce an admission control for group of streams.

Finally, we validate Zeus and we show that the Zeus architecture provides better real-time guarantees with concurrency and dependency constraints than the standard Linux kernel. We also show that it is crucial to consider dependencies and concurrencies in order to achieve QoS service guarantees and avoid increased delays caused by bouncing effects due to the lack of Instream-Outstream Synchronization.

Chapter 6

Prometheus: Streaming as a Service Kernel

3D TI systems make use of Content Delivery Gateways (CDG) to transfer content from input devices to remote output devices over the Internet. Apart from streaming, CDG need to provide additional processing of streams including overlay routing, bandwidth management, QoS provisioning, synchronization, and monitoring. Processing of streams at each CDG is dependent on the type of activity at the 3D TI. For example, Multiplayer Online Gaming might require authentication and collision detection, while a Physiotherapy session might require encryption to ensure doctor-patient confidentiality. Additionally, the presence of multiple correlated sensors interacting in a 3D TI system requires support for groups of *large scale correlated multi-streaming* (i.e., Bundle of Streams) as an inherent functionality. Moreover, the rapid change of hardware in 3D cameras and sensors, the lack of standard stream formats and the heterogeneity of devices present the need of enabling Universal Access to large range of heterogeneous streaming devices. Finally, the distributed nature of I/O devices in 3D TI Systems require consistent resource naming to sites and devices that allows to preserve the geographical location awareness and context of each group of devices and streams associated with each particular 3D TI site.

Streaming protocols like DASH [65], defined in the ISO/IEC 23009-1 standard and RTCWeb [66], submitted as a draft to IETF, provide mechanisms for streaming audio and video over the internet, however they do not provide mechanisms to provide policy enforcement to achieve QoS in 3D TI Systems, also the protocols does not consider the problem of preserving location-context of an stream in 3D TI Sites

Gateway architectures like those proposed in [55], [56], [57], [58], [59] provide streaming and processing over Local Area Networks (LAN) using Ethernet, Bluetooth and 802.11. However, processing in this approaches is limited to relaying, multiplexing, translating, and managing local resources only. These approaches are general architectures for home and sensor networks and do not consider activity dependent processing, neither they consider the dependencies created by time and space correlated streaming in 3D TI Systems.

Streaming middleware frameworks like CoolStreaming [63], Nizza [17] and [64] provide support for processing and dissemination of streams. However, they lack support for management of groups of time and space correlated streams as

required by 3D TI Systems.

Finally, proprietary gateways for Teleimmersion like HP Halo [4], Cisco Telepresence [3], and Technicolor are tailored to cater closed applications and therefore they are unsuitable for the multimodal non standard interfaces of 3D TI Systems. While, some interoperability standards between 3D Telepresence systems have been proposed, like the CLUE data model (Controlling Multiple Streams for Telepresence) [67], this standards are limited as they only consider audio and video and they do not consider multimodal sensors. Also, this standards do not consider mechanisms to provide Quality of Service neither they considers diverse physical activities as required by more advanced 3D TI Systems.

6.1 3D TI Streaming Challenges and SAS Goals

To address the challenges in 3D TI Streaming, we envision a novel paradigm, *Streaming as a Service (SAS)* to model correlated multi-streaming service, where groups of time and space correlated streams, also called *Bundle of Streams*, are first class objects. We propose a SAS-based, generalized, distributed service kernel, *Prometheus* to setup, process, and control bundles of streams through a unified interface for diverse end-devices and User-controlled run-time functions that operate over frames, streams and bundles. This streaming model is driven by the challenges in 3D TI Streaming. In this section we discuss in detail each of the challenges in 3D TI Streaming and how our SAS model addresses them:

- *Correlated Multi-Stream Support:* 3D TI Systems are characterized by the dissemination of groups of streams called *Bundle of Streams (BoS)* sharing high spatial and temporal correlations. These bundles interact in synchronous and soft real-time manner. The current streaming protocols like RTP/RTCP, SIP and RTSP do not take into account efficiently the spatio-temporal dependencies among large sets of streams. SAS provides support to correlated multi streaming by incorporating groups of time and space correlated streams, called Bundle of Streams [27] as first class objects. In SAS, Bundles of Streams are processed and disseminated as a group throughout the entire overlay routing topology until they arrive at the remote sites, where the Bundle of Streams is rendered at the displays and sensors to recreate the remote 3D TI site.

- *Distributed Nature of 3D TI I/O Devices:* A 3D TI System is comprised of multiple geographically distributed sites connected through Internet2, in which Content Delivery Gateways must disseminate groups of correlated streams from devices across this geographically distributed sites. Therefore, 3D TI systems require consistent resource naming to sites and devices

in 3D TI Systems that allows to preserve the geographical location awareness and context of each group of devices and streams associated with each particular 3D TI site. SAS addresses this challenge by providing *Hierarchical Uniform Naming* to stream groups of correlated streams so that individual streams can be identified within a specific room and session in a 3D TI System.

- *Non-Standard Heterogeneous I/O devices:* 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 by using socket-level network tunneling as a well-defined interface to the end-devices. In SAS, varied types of streaming devices with different standards seamlessly connect and stream the data.

- *Activity-dependent Stream Processing:* Stream Processing in 3D TI Systems is highly dependent on the type of activity, as different activities require different processing algorithms and functions. For example, Multiplayer Online Gaming might require authentication and object collision detection, while a Physiotherapy session might require encryption to ensure doctor-patient confidentiality. To address this challenge, SAS follows the principle of *Separation of mechanism and policy* [100], 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. SAS provides two types of run-time functions on streams: system-defined functions like rate control, congestion control, and multi-stream synchronization and user-defined functions like compression, encryption, and view management. These functions can be requested in an on-demand basis.

Therefore, the goal of the SAS is to foster bundles of streams needing correlated multi-streaming support, universal access across multimodal devices, and user and activity controlled run-time functions in 3D TI Systems. To realize the SAS paradigm, we present Prometheus, a set of real-time integrated streaming and processing services that address the challenges of 3D TI by ensuring the properties of the *SAS paradigm* at run-time.

As part of our solution we provide:

1. Formalization of the Streaming as Service model.

2. Hierarchical Uniform Naming for 3D TI Systems that allows to preserve the location-context information of streams with respect of which room and session they belong to.

3. A streaming protocol for real-time data delivery (S-RTP) that uses our Hierarchical Uniform Naming to stream groups of correlated streams (i.e., Bundles of Streams).

4. Design of the Prometheus streaming framework that provides streams and bundles as first class objects, unified interface for multimodal end-devices, user-controlled run-time functions over streams and bundles and integrated management for Bundle of Streams.

6.2 Prometheus Architecture

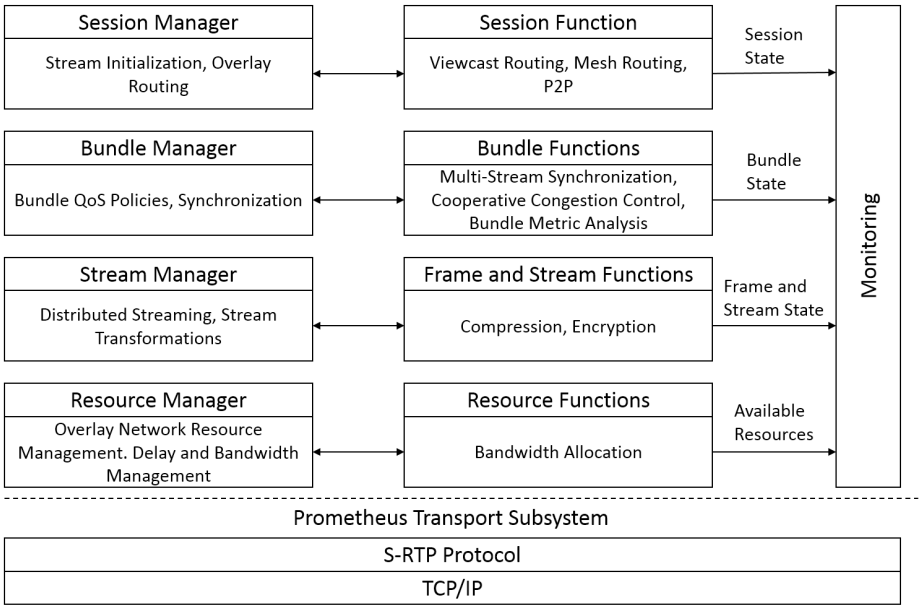


Figure 6.1: Prometheus Framework

In Prometheus, the SAS properties get implemented as management, run-time, and monitoring entities in the session subsystem on top of a transport subsystem. Since streams are first class objects in Prometheus, each of the entities keeps track of and controls streams and stream derivatives (e.g., bundles, frames). Figure 6.1 shows the layout of various entities over the transport subsystem. In this section we describe each of this entities.

6.2.1 Management Entities

The management entities manage 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 Stream Initialization, Session Management and Routing. It takes management decisions and provides mechanisms to load session level functions like overlay routing.

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.

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 run-time 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.

6.2.2 Run-time Entities

The Run-time 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 system-admins or the end-users of Prometheus. Examples of run-time entities at each level are shown in Figure 6.1.

6.2.3 Monitoring Entity

Prometheus 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.

6.2.4 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 3D TI specific S-RTP protocol which adds semantic information (used by managers) like stream type, functions requested, device addressing, and streams in same bundle.

6.3 Streaming Protocol in Prometheus

Prometheus is realized through a set of multiple distributed SAS gateways and SAS interfaces as shown in Figure 6.2. SAS gateways take on the responsibility of hosting the distributed instances of the Prometheus Kernel and the SAS interfaces (SASI) provide the connectivity between the end-devices and the kernel of Prometheus. We assume that all gateways and end-devices can be connected to each other via the Internet. Figure 6.3 shows the end-devices, SAS interfaces, and the functional placement of the entities of the Prometheus kernel in a gateway. The streaming algorithm is as follows:

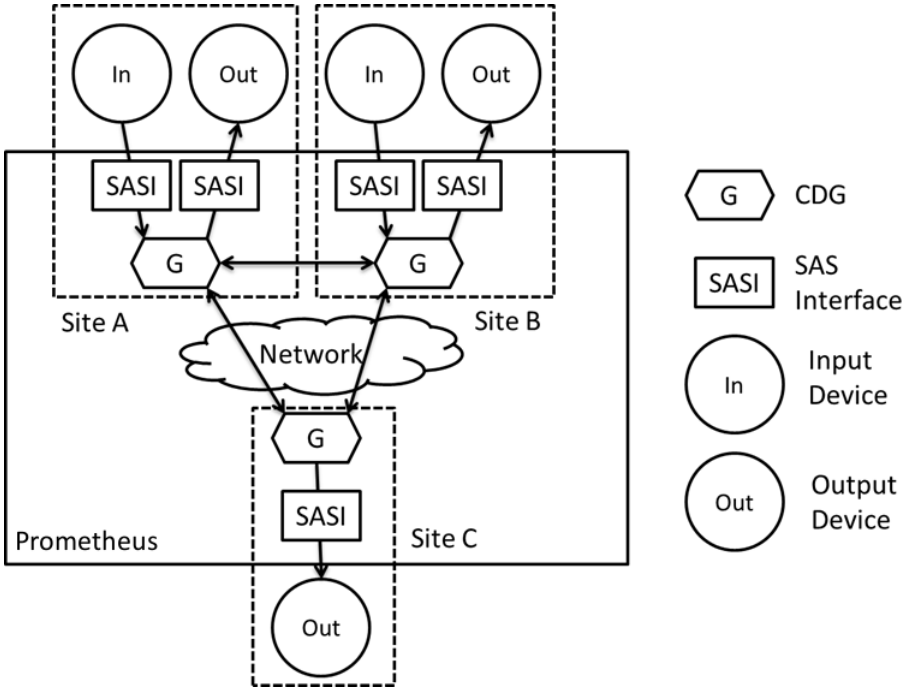


Figure 6.2: Distributed Components of Prometheus

1. Stream Initialization: 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

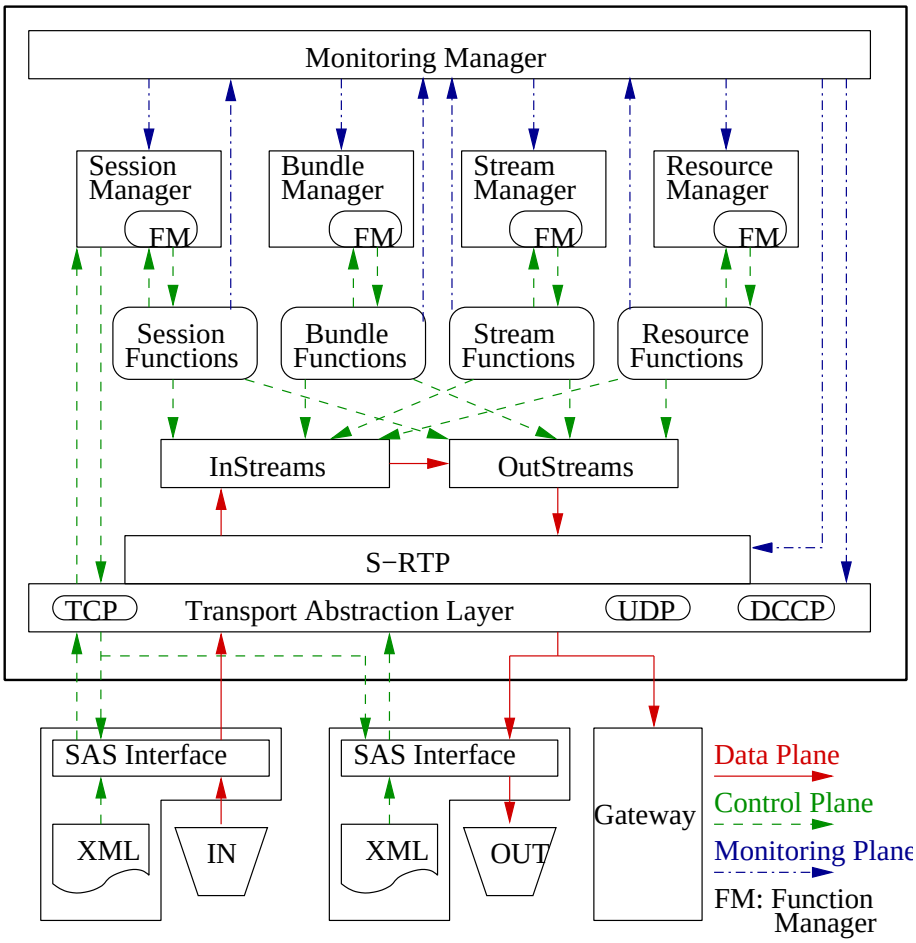


Figure 6.3: Prometheus Data, Control, and Monitoring Planes

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. Run-time 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 Manager applies policies for bandwidth management and congestion control. It must be noted that streams pass through all these functions only when the user demands them. Thus, no extra overhead is incurred unless some

functions are specified. This ensures fastest delivery of streams.

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.

6.4 Prometheus Interface and Function Managers

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

6.4.1 SAS Interface

The device interface with Prometheus provides universal open access and faces the challenges of multiple non-standardized stream formats of end-devices. This challenge severely affects 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 Prometheus kernel using configuration mechanisms to specify the formats at run-time. Thus, Prometheus realizes four concepts:

1. End-to-End Tunneling.
2. Device Stream Specification.
3. Semantic data propagation through S-RTP.
4. Service Negotiation.

End-to-End Tunneling

The idea behind the Streaming as a Service model is that end-devices should interact agnostically with Prometheus (i.e., the end-devices do not know if they are communicating via Prometheus). 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 Prometheus. This effectively creates an application layer tunnel between the streaming devices.

The assumption behind using socket API is that the end-devices in 3D TI Systems 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

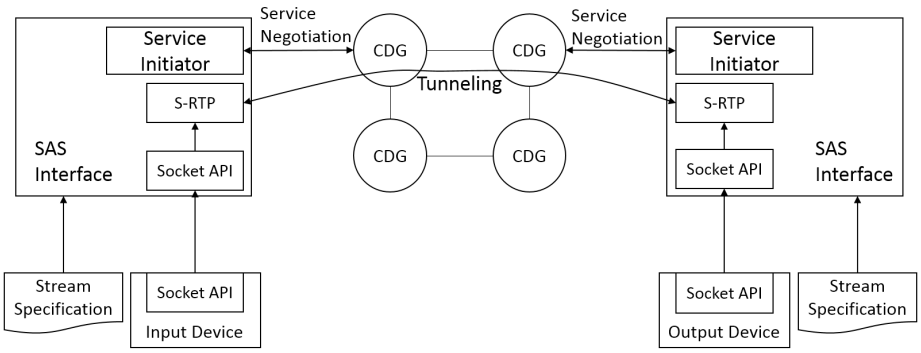


Figure 6.4: Socket Interface and Tunneling

from the input devices and send it via Prometheus 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 Prometheus kernel. Figure 6.4 shows the socket interface and the tunnel. From a networking point of view, SAS Interface and Prometheus 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 Prometheus.

Device Stream Specification

In order to apply functions on streams, Prometheus 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: A Device Specification containing general metadata about the device and a Stream Specification containing stream format. The Device Specification is explained in Chapter 7

```
<STREAM SPECIFICATION>
   <PACKET_FIXED>
     <HANDSHAKE> ON </HANDSHAKE>
     <PACKET_SIZE> 140 </PACKET_SIZE>
     <PACKET_COUNT> 1 </PACKET_COUNT>
   </PACKET_FIXED>
   <PACKET_VARIABLE>
     <HANDSHAKE> OFF </HANDSHAKE>
     <HEADER_SIZE> 10 </HEADER_SIZE>
     <HEADER_OFFSET> 6 </HEADER_OFFSET>
     <DATASIZE_TYPE> 4 </DATASIZE_TYPE>
     <ENDIANNESS> NETWORK </ENDIANNESS>
     <PACKET_COUNT> -1 </PACKET_COUNT>
   </PACKET_VARIABLE>
</STREAM SPECIFICATION>
```

Figure 6.5: Stream Specification

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*. Prometheus 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 6.5 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.

The Stream Specification is the key to provide Universal Access to a large scale of streaming devices as it allows to abstract the device specific streaming protocol. This stream specific protocol information is then embedded in the S-RTP protocol and disseminated along all the CDG along the streaming routing path of the stream.

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 3D TI 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.

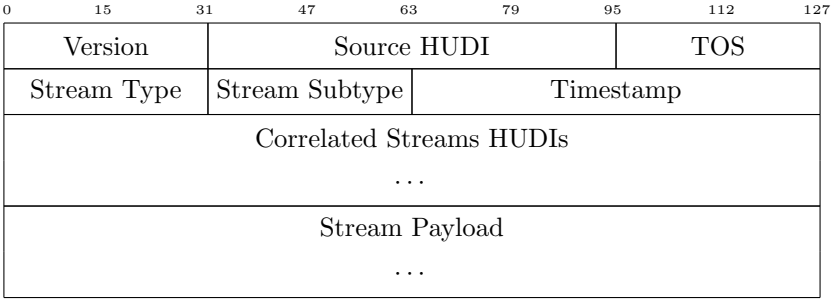


Figure 6.6: S-RTP Header Specification

The structure of the S-RTP packet is shown in Figure 6.6. The packet first specifies the version of the S-RTP protocol followed by a 64 bit unique stream identifier, referred to as the Hierarchical Unique Device Identifier (HUDI). The unique identifier uses a hierarchical addressing scheme composed of: *Session ID* that uniquely identifies the TI session, *Site ID* that identifies the TI Site and *Device ID* that uniquely identifies the plugged-in device. The structure of the HUDI is shown in Figure 6.7. The HUDI provides consistent resource naming to sites and devices in 3D TI Systems that allows to preserve the geographical location awareness and context of each group of devices and streams associated with each particular 3D TI site.

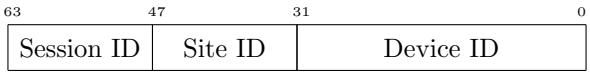


Figure 6.7: Hierarchical Universal Device Identifier

The Type of Service (TOS), a 64 bit flag vector, specifies the requested functions, the state information about functions that were applied along the route in Prometheus, and a Handshake bit to specify Handshake packet. The stream type and subtype together form a tuple to uniquely identify the type (e.g., 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, timestamp of packet creation and the device payload.

Stream Initialization and Service Negotiation

After reading the stream specification and constructing an S-RTP packet, the SAS Interface at the joining end-device initiates a session with Prometheus. The Session Manager in Prometheus handles the stream initialization and service negotiation tasks. Remote procedure calls (RPC) and marshalling are used between the SAS Interface and Prometheus. Our contribution is that the Prometheus allows dynamic pluggability of different 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 subtype, expected bandwidth usage), and the services requested (encryption, compression, congestion control). Upon receipt of the *JOIN* message, Prometheus verifies whether it can support services requested, and if so, opens required data ports and returns an *ACK* containing the ports. Prometheus renegotiates if it does not support any of the services with the SAS Interface. The Session Manager in Prometheus 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 Prometheus 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.

6.4.2 Prometheus Function Manager

In order to provide run-time stream-processing functions, i.e., user controllable functions, each manager in Prometheus implements a Function Manager (FM) as shown in Figure 6.3. The Function Manager is responsible for implementing mechanisms to control and schedule functions on bundles, streams, and frames. Run-time functions are a key feature in our solution as they allow activity-driven processing as it is required in 3D TI Systems. They allow the participants to dynamically change the set of processing functions for a group of stream based on the activity that they are engaging.

To support this user-defined run-time functions, Prometheus divides the execution plane in two spaces: *End-User Space* and *System Space*. The *End-User Space* is the space where processing Functions execute, while all the other functions and resource management remain in the System Space. Each processing Function has a separate address space for each stream to ensure proper resource management and policy enforcement. The *System Space* is responsible of providing context information about the Stream, Bundle or Frame to the End-User Space. It also provide resource management and protection across other processing Functions.

New functions to be added to Prometheus 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. Execution Flow is driven by the Scheduler in the System Space. The Scheduler is responsible for selecting the next Function and transferring control to the Function in the User-Space. Figure 6.8 shows the architecture of the Function Manager. The Figure shows the Function Manager with two streams; Stream 1 is being processed by two functions while Stream 2 is being processed by one function.

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

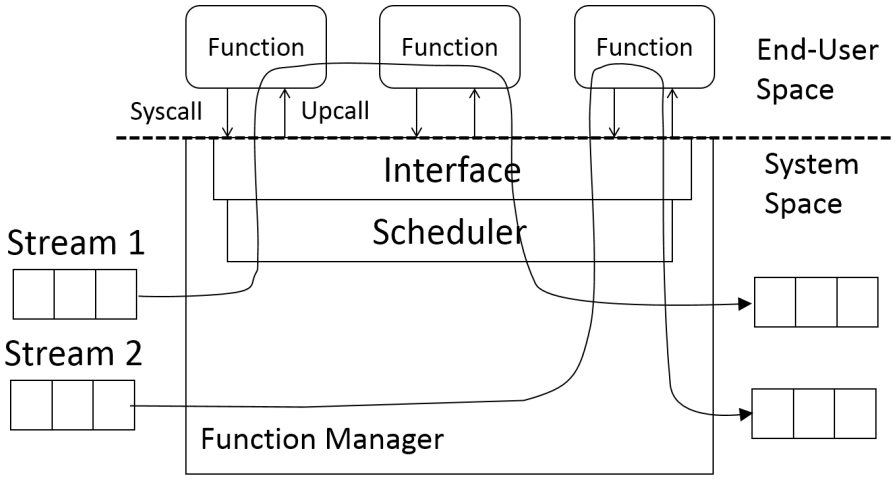


Figure 6.8: Prometheus Function Manager

at each level of data abstraction and load user-specific functions to implement these mechanisms. This ensures high extensibility of services in Prometheus.

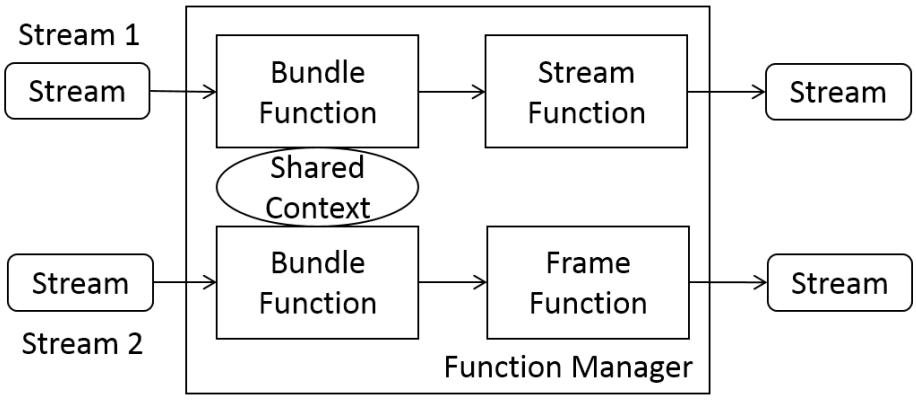


Figure 6.9: Prometheus Function Manager Data Plane

Prometheus divides Processing Functions into 3 types:

1. Frame Functions: These functions operate only over a single frame of a single stream and their context is not preserved across each invocation (i.e., callback). Example of this functions are Encryption and Compression.

2. Stream functions: These functions operate over the frames of a single stream and they preserve their context across each invocation. Example of this functions are QoS algorithms (e.g., Random Early Drop or Token Bucket).

3. Bundle Functions: These functions operate at the Bundle of Streams level. They operate over the frames of each of the streams forming a bundle. Therefore, their context must be shared across several Streaming

Instances. One example of these functions is Multi-Stream Synchronization [93].

Figure 6.9 shows the execution overview for each Processing Function type.

6.5 Experimental Evaluation

For the evaluation of Prometheus, we used a 3DTI System composed of two sites. For visual purposes, we use Bumblebee2 Stereo Cameras as input devices and one display as an output device in each site. Each stereo camera produces a variable 3D video stream with bandwidth demand ranging between 1 to 2Mbps. Each site also contains audio input and output devices to create synchronized audio for collaborative activities. The experiments are all performed on Dell Precision 690 with dual Intel Xeon processor and 2GB of RAM running Linux 2.6.20.

We perform two scenarios to evaluate our system. In *scenario 1*, we measure the overhead in terms of delay and CPU usage of the *Prometheus* while streaming to a local device. In *scenario 2*, we measure the overhead (delay and CPU usage) of Prometheus while streaming to a remote device. For both scenarios, to stress the processing and bandwidth power of the system, we vary the number of devices (streams) from 2 to 24 (by varying the number of camera streams).

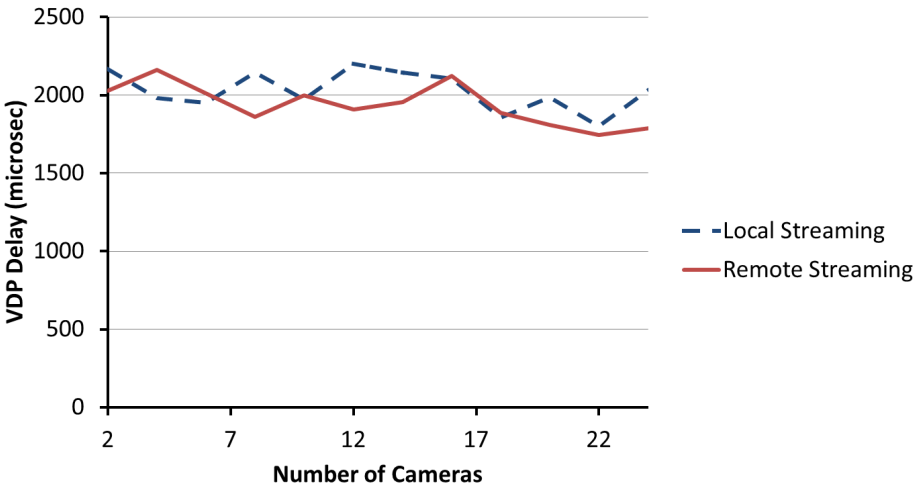


Figure 6.10: Prometheus component delay streaming to local and remote nodes

Component Delay Overhead: We evaluate the total component delay overhead added by the Prometheus wherein total delay is the difference between the entry time of a frame and the exit time of that frame at the CDG. Figure 6.10 show that the average total delay for a local node streaming is less than 2.5 milliseconds even for 24 concurrent streams, and increases minimally for streaming to remote nodes. This shows that Prometheus is suitable for the interactive

applications found in 3D TI Systems while providing a universal interface with ubiquitous access and location transparency.

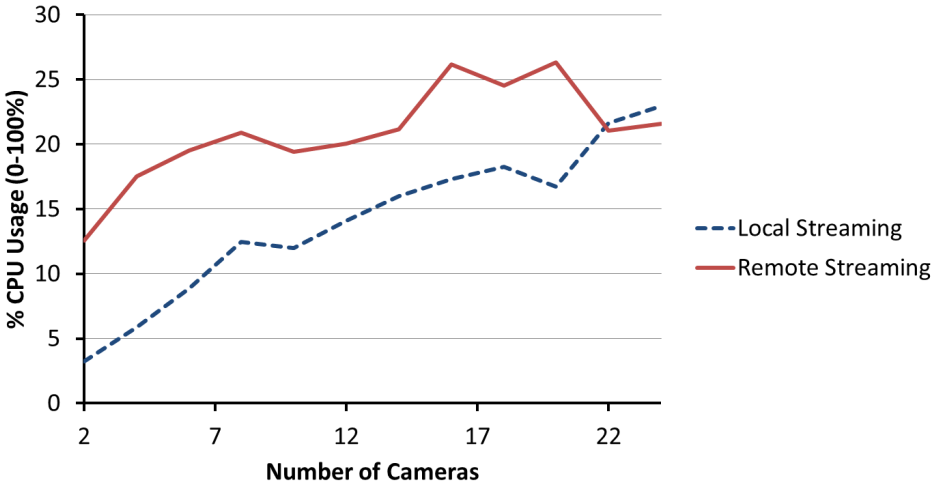


Figure 6.11: Prometheus CPU usage for streaming to local and remote nodes

CPU Overhead: It is important for Prometheus to scale in terms of CPU demands as large number of end-devices are added to the system. As shown in Figure 6.11,the average CPU overhead ranges between 2% for 2 streams to 20% for 24 streams for local node streaming (scenario 1). For remote streaming, the average CPU requirement only increases to 25% for 24 streams (scenario 2). This emphasizes that Prometheus demands low CPU even when large number of high-bandwidth streaming-based device are connected to it.

Message Overhead: Prometheus adds S-RTP header on the data packets and uses Google Protocol Buffers for marshaling S-RTP frames. For the current implementation of Prometheus, only a fixed cost of 22 bytes per frame is incurred as an S-RTP header. The Google Protocol Buffer layer only adds 4 bytes to the header. Thus, a total of 26 extra bytes per frame over the typical frame size ranging from 2KB to 30KB for 3D-video frames.

Prometheus Stream Initialization Time: We evaluate the initial delay added when a device establishes a connection with Prometheus. For this experiment we measured the initial delay overhead over several initial connections. Figure 6.12 shows a trace of 10 different connections with Prometheus. The delay is measured with devices connected to the local node. As shown the maximum delay incurred is 420 microseconds, while the average is 360 microseconds.

Average CPU Usage: We measure the CPU usage of Decima and Prometheus at the CDG during steady streaming between two sites using one Bumblebee Stereo Camera to one rendering device in the remote site. We compared this value with the average CPU usage of Yang et al. [7] and directly connecting a camera to a rendering device with no middleware (i.e., Device-to-Device). Table

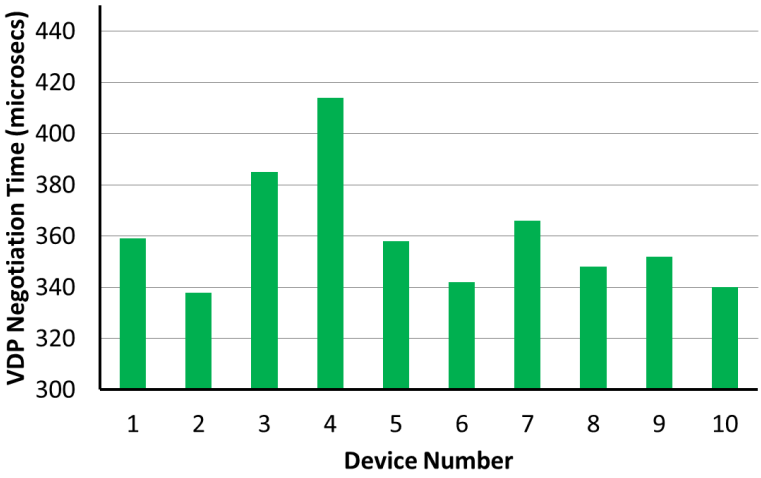


Figure 6.12: Prometheus Stream Initialization Time

	Prometheus	**Yang et al.**	**Device to Device**
Avg. Streaming CPU Usage	26.5%	2.7%	Device Specific
Avg. Streaming EED	37.2 ms	35.3 ms	15.3 ms
Interfacing Code	88 XML lines	863 C++ lines	Hardcoded
Multi-Stream Support	Audio and Video	Video only	Video only
Multi-Site Support	Yes	Yes	No

Table 6.1: Comparison of Prometheus with other systems

6.1 shows the values for the 3 systems.

Average Streaming End-End Delay: We measure the End-to-End Delay (EED) incurred by Decima and Prometheus for streaming between two sites using one Bumblebee Stereo Camera to one rendering device in the remote site. We compare this value with the average EED of Yang et al. [7] and directly connecting a camera to a rendering device with no middleware (i.e., Device-to-Device). Table 6.1 shows the values for the 3 systems.

Comparison with Existing Systems: As part of our validation, we present a comparison of our TI System using Prometheus with existing TI Systems. Table 6.1 shows the comparison of Prometheus, the system proposed by Yang et al. [7] and Device-to-Device using no additional virtual layer. The features compared are: average CPU usage while streaming, if the system support multiple streams (i.e., Multi-Stream), if the system support multiple sites (i.e., Multi-Site). Also, we compare the number of additional lines of code the developer must write as interface to the middleware system for streaming from the device (i.e., Interfacing Code).

6.6 Conclusion

As part of our contribution, we introduce the Streaming as a Service as a flexible, configurable and scalable model for streaming of multimodal devices as found in 3DTI. In the SAS concept, real-time data streaming is provided between input and output devices as a transparent layer. Moreover, access to disseminating infrastructures is provided through a universal interface in which multimodal devices require no source code modification to interface and instead they provide a specification about their streaming protocols. Our paradigm introduces the concept of groups of time and space correlated streams (i.e., Bundle of Streams [27]) as first class objects.

We also introduce data streaming protocol (S-RTP) based on hierarchical uniform naming for devices, sites and session in 3D TI Systems. Our hierarchical uniform naming solves the problem of location-context preservation in 3D TI and Telepresence Systems as it allows to uniquely associate a device with a site and a session. We believe that this naming should have an impact in future standards and in currently proposed standards for interoperability between 3D TI Systems (e.g., Clue [67]).

We validate this model through the implementation of Prometheus, a streaming and processing framework for groups of time and space correlated multistreams. Prometheus supports the activity driven processing of 3D TI Systems by incorporating user-defined processing functions over Bundles of Streams. Our experiments in a real 3D TI System indicate that Prometheus is successful at providing the streaming and processing requirements of 3D TI Systems without significant delay and bandwidth overhead.

Chapter 7

Decima: Dynamic I/O Device Management

3D TI system is composed of a large scale of multimodal streaming devices that present several challenges to Device I/O Management in these systems. The Diversity and contingency of activities present the challenge of the *dynamic configuration* support for I/O devices. Non-Standard Heterogeneous I/O devices, the rapid change of hardware in 3D cameras and sensors and the lack of standard stream formats present the need of enabling *universal access* to these I/O devices. The distributed nature of devices in 3D TI Systems presents the challenge of providing *consistent resource naming* to sites and devices in 3D TI Systems that allows to preserve the *geographical location awareness and context* of each group of devices and streams associated with each particular 3D TI site. Finally, dynamic view changes, in which each site requests access to different devices (e.g., front or side cameras) require *virtualized access and control* of devices.

Videoconferencing applications, like Skype and LiveMeeting, access devices through low-level device management I/O systems like USB [75] or Firewire and access standard multimedia formats using services like Video4Linux and Windows Media. Each user controls its own devices and the application is responsible for sharing the content across multiple sites. However, this is insufficient for 3D TIs, where devices are *remotely controlled by the trainer* participating in the asymmetric activity. Asymmetric activities are significantly different when compared with the type of interactions in Skype and other videoconferencing applications where each of the participants symmetrically controls the devices.

On the other hand, application level distributed I/O systems allow high-level multiplexed access to devices like Server Message Block (SMB/CIFS) [80], PulseAudio [30] and iSCSI [74]. However, existing application level distributed I/O systems do not provide adequate semantic support for groups of streams, neither do they support multi-modal interfaces.

Additionally, service gateway platforms like OSGi [60] and others (e.g., [55], [56], [57], [58]) allow distributed device management in an open access interface that supports multimodality. However, OSGi has a very low-level interface and it is designed for small sensors and automation, making it very cumbersome to build complex TI systems that require multi-correlated streaming.

In this paper we present Decima, a holistic, virtualized, configurable and

scalable, distributed I/O management subsystem for 3D Teleimmersive Networks. Decima allows dynamic, seamless and universal access to a large-scale of heterogeneous distributed stream-based I/O devices. Decima provides device-site management support for time and space correlated groups of interactive streams.

In summary our contributions are:

1. Device and Resource naming protocols, based on a Hierarchical Uniform Device Identifier for 3D TI Systems that address the challenge of preserving the location-context information of devices and streams with respect of which room and session they belong to.

2. A dual virtualization architecture that addresses the challenge of enabling seamless dynamism of multimodal devices, i.e., hides against changes in hardware interfaces in multimodal devices (e.g., if the 3D camera used changes from Bumblebee to Kinect) and allows seamless dynamism of activities, i.e., enables universal interface of activities to distributed I/O, without any concerns of the underlying software changes (e.g., if user activity changes from walking to sitting).

7.1 Overview of Decima

Decima is a holistic, virtualized, configurable and scalable, distributed I/O management subsystem for 3D Teleimmersive Networks. Decima allows dynamic, seamless and universal access to a large-scale of heterogeneous distributed stream-based I/O devices. Decima provides device-site management support for time and space correlated groups of interactive streams.

Decima uses an end-to-end approach to device management and control, i.e., activities in Decima connect and control remote devices in the same way as if they were local. Decima virtualizes the device control and management to provide location transparent and hardware independent access across multiple sites in the 3D TI System, as required by asymmetric activities. Figure 7.1 shows the distributed virtualized I/O management of Decima. The example shows a 3D TI System with 3 sites with 3D cameras at each site are managed and controlled by the trainer at Site 1. Device Commands from Site 1 are used to control the remote and local cameras transparently from their location within the 3D TI network or independently from the type of underlying camera hardware used at each Site.

7.2 Architecture of Decima

Decima uses a *Dual Virtualization* composed of a *local virtualization layer*, referred to as the Device Interface Layer (DIL), that copes with the dynamism of hardware implementations of devices; and a *remote virtualization layer*, referred

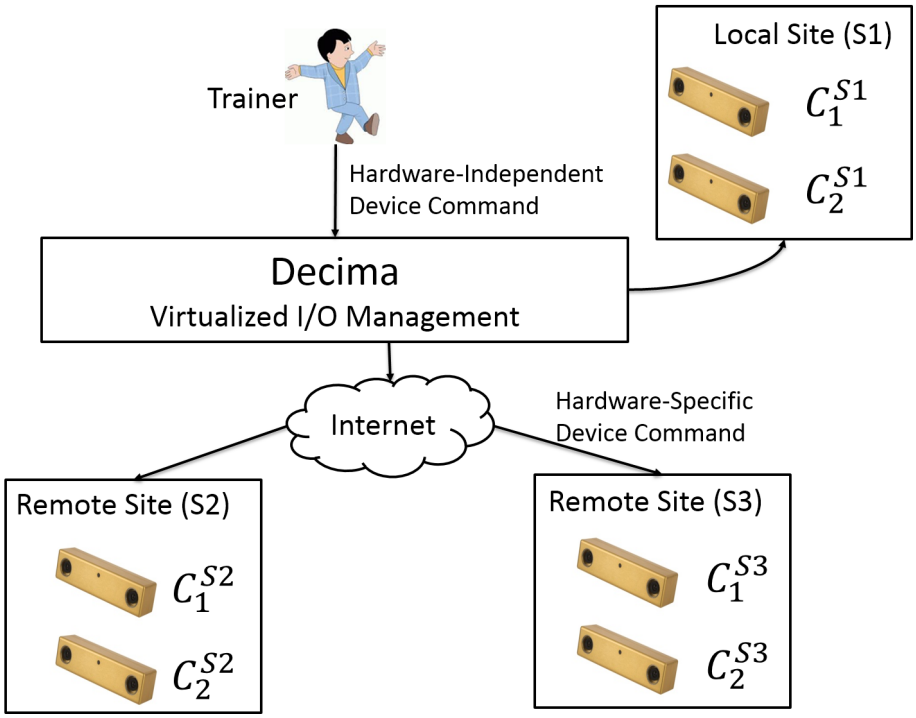


Figure 7.1: Distributed Virtualized I/O Management of Decima

to as the Distributed Device Control Layer (DDCL), that copes with the dynamism of the distributed activity in TIs. Figure 7.2 shows the dual abstraction architecture of Decima.

7.2.1 Distributed Device Control Layer

The Distributed Device Control Layer (DDCL) represents the *remote virtualization* layer of Decima. It enables seamless control of devices for any TI activity, independent if the devices are local or remote, hence responds efficiently to changes and demands coming from diverse activities. The DDCL layer spans across all the sites in the 3D TI System. The DDCL is composed of three main components: *Dynamic Device Configuration Service*, *Device Controller* and *Device Control Service and Protocol*. Figure 7.3 shows the architecture of the DDCL. We explain each of the components below.

Dynamic Device Configuration Service: In Decima, each site in the 3D TI System must run a Dynamic Device Configuration Service (DDCS) to manage device connectivity in 3D TI Systems. This service provides registration and device name assignment after the devices are plugged in by the user. When the device gets plugged in, the *Device Interface Layer* sends a registration request to the DDCS. The registration request contains information about the device including the *Device Class*, the device manufacturer and model, a bitmap listing the capabilities (e.g., real-time capability for camera devices and non-real-time

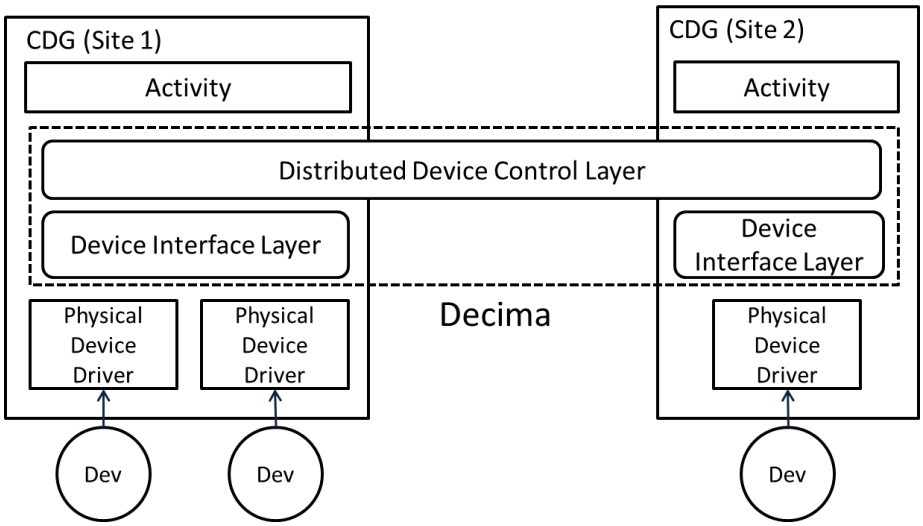


Figure 7.2: Dual Abstraction Architecture of Decima

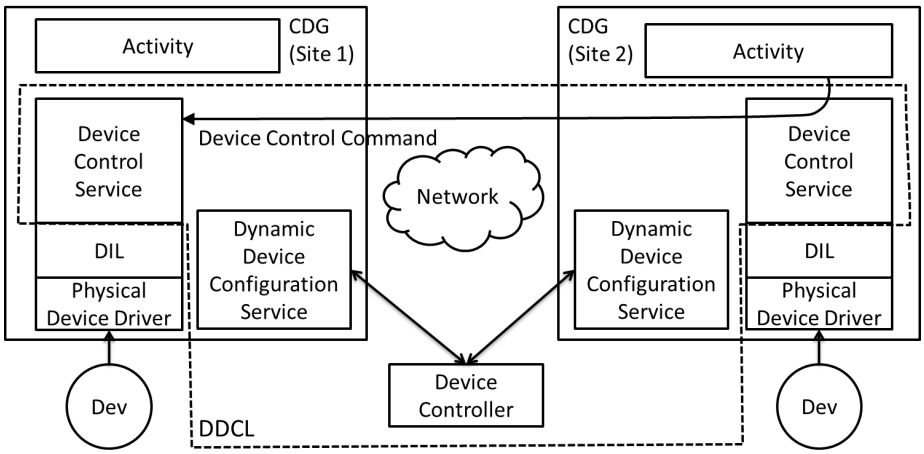


Figure 7.3: Distributed Device Control Layer architecture

capability for storage devices) of the device and the location information of the device.

The location information of the device contains the IP address and control port associated with the Ethernet based device. In the case of devices connected through a local bus, the location information contains the IP address of the CDG in the local site. In the case of devices connected through a local bus, the location information contains the IP address of the CDG in the local site. However, the table might contain more specific geolocation information from future devices (e.g., GPS information or other localization information acquired via indoor localization techniques).

After receiving a registration request from the DIL, the DDCS will perform name assignment on the device by associating the device with a Hierarchical Unique Device Identifier (HUDI). The format of the HUDI is described in Chap-

Field	Size	Description
HUDI	64 bits	Device unique identifier
ip_address	32 bits	IP address of the device driver
control_port	32 bits	Port number to send Device Control Commands to the device driver
payload_type	32 bits	Type of data produced by the device (e.g., VIDEO, AUDIO)
payload_subtype	32 bits	Data Format (e.g., YUV12, MPEG-2, SCSI)
stream_direction	32 bits	Input or Output Device
manufacturer	variable	String identifying the device manufacturer
model	variable	String identifying the device model
device_class	32 bits	Virtual Device Class
capabilities	64 bits	Bitmap of Capabilities supported by the device
status	32 bits	Status information of the device
device parameters	variable	Device dependent parameters (e.g., frame rate, resolution, color depth, pixel format)

Table 7.1: Description of Device Table fields.

ter 6. Streams from this device will also be assigned the same HUDI. Therefore, the HUDI serves as a unique identifier for both streams and devices. After assigning the HUDI to the devices, the DDCS adds an entry for the device into a local Device Table that keeps track of all the devices currently connected to the site. When a device is unplugged the DDCS will release any assigned HUDI and will remove the device from the Device Table.

Device Controller: The *Device Controller* aggregates information about all the devices connected in the 3D TI System. Its purpose is to provide a centralized repository containing information about the devices of all the TI sites (i.e., status of the device, model). After the registration and name assignment are complete at the local DDCS, the DDCS must send an update notification to the *Device Controller* in the system containing information about the new device in the 3D TI System.

Remote CDGs periodically query the Device Controller for information about new devices. This information is used to notify the activity that a new device is available and can be used by the remote nodes. Similarly when a device is unplugged, the Device Controller is notified by the DDCS and the Device Controller removes the device entry from the list of available devices.

Device Control Service and Protocol: The Device Control Service is responsible for receiving commands from the remote CDGs and forwarding them to the local *Device Interface Layer*. The Device Control Service runs a protocol between its peer entities to forward *Device Control Commands*. It means that if an activity at Site 1 wants to send a device control command to a device at Site 2, it contacts its own Device Control Service on Site 1 which forwards the device control command to Device Control Service at site 2 using the Device Control Protocol between them.

The Device Control Protocol header at its Application Protocol Data Unit (APDU) captures the semantics of a generic device control command. The size of this header is 190 bits. The protocol first specifies the version followed by the source HUDI, the HUDI of the device that will be accessed (e.g., destination HUDI) followed by the Device Control Command. The protocol first specifies the version followed by the source HUDI, the HUDI of the device that will be accessed (e.g., destination HUDI) followed by the Device Control Command. Figure 7.4 shows the structure of the Device Control Protocol.

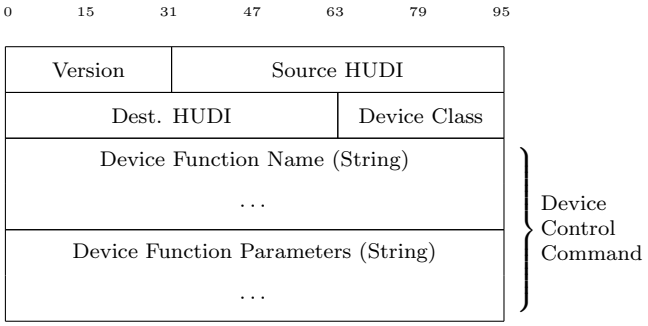


Figure 7.4: Device Control Protocol specification

7.2.2 Device Interface Layer

The Device Interface Layer represents the *local virtualization* layer of Decima. It hides changes in the hardware implementation of devices in TIs. The DIL is composed of two main components: *Device Class Driver* and *Device Stream Specification*. Figure 7.5 shows the architecture of the DIL. Below we explain the overall interaction of the components in the system and in the following subsections we provide a detailed description of each component.

Device Class Driver: The Device Class Driver serves as interface that allows access to the device independent from the specifics of the hardware implementation. It is responsible of translating device independent requests into physical device driver dependent requests.

In Decima, each type of device gets assigned a different *Device Class Driver*. Each *Device Class Driver* provides a specific interface that abstracts all the operations common to all devices of the same type. All the communication with the device must go through this interface.

Figure 7.6 shows the device class interface for the `CAMERA` device class. Each of the operations of the stereo camera is mapped into one of the callback functions listed in the *Device Class Driver* interface. When a Device Control Command is received by the Device Control Service it gets mapped by the Device Class Driver into one of this callback functions. The callback function is responsible for translating the device independent request into physical device driver dependent requests.

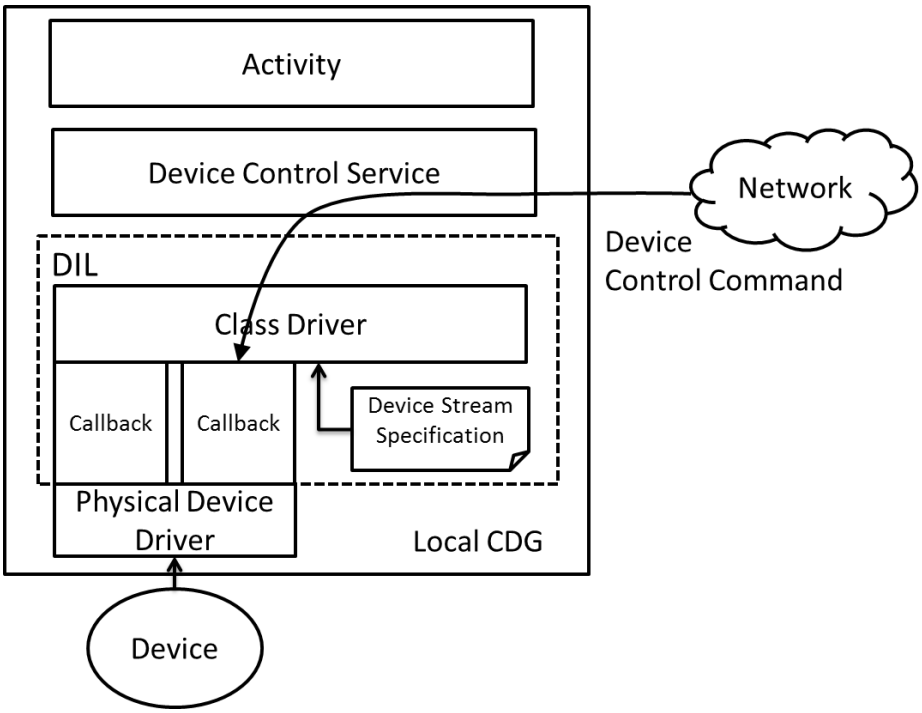


Figure 7.5: Device Interface Layer architecture

Each Device Class Driver implements a `Query()` function. This function can be used by the activity to obtain information about the device including the device status, manufacturer, model and the type of streaming payload of the device. This information can be obtained through the physical device driver if supported by the device or through the Device Stream Specification.

```
class ICameraDriver
{
  UINT32 deviceTurnOn(UINT64 SourceSiteID);
  UINT32 deviceTurnOff(UINT64 SourceSiteID);
  UINT32 startRecording(UINT64 SourceSiteID);
  UINT32 Query(STRING key, STRING &value);
  UINT32 stopRecording(UINT64 SourceSiteID);
  UINT32 setSetting(STRING key, STRING value);
}
```

Figure 7.6: CAMERA device class interface

The *Device Control Command* sent by the activity matches the syntax structure of the *Device Class Driver* interface. This interface is fixed for each device class and therefore our design does not need proxy/stub code, in contrast to RPC or CORBA. This simpler design is tailored for device control and management in the closed 3D TI systems that does not require arbitrary object serialization, process activation or other complex features needed in general purpose remote procedure invocations.

Device Specification: The Device Specification allows the DIL to obtain device semantics without relying on specific hardware implementations. The Device Specification includes the type of the device (device class). This value is used by Decima to load the proper Device Class Driver. It also specifies additional semantic information about the device including the manufacturer of the device, if the device is an input or output device and the type of streaming payload (e.g., video, audio, sensors). Figure 7.7 shows the Device Specification of a Pointgrey Bumblebee camera used by Decima.

```
<DEVICE SPECIFICATION>
  <DEVICE CLASS>CAMERA</DEVICE CLASS>
  <PAYLOAD TYPE>VIDEO</PAYLOAD TYPE>
  <MANUFACTURER>POINTGREY</MANUFACTURER>
  <MODEL>STX</MODEL>
  <STREAM DIRECTION>OUTPUT</STREAM DIRECTION>
  <DEVICE CONTEXT>
    <FRAME RATE>20</FRAME RATE>
    <RESOLUTION>640x480x200</RESOLUTION>
    <COLOR DEPTH>8</COLOR DEPTH>
    <PIXEL FORMAT>RGB</PIXEL FORMAT>
  </DEVICE CONTEXT>
</DEVICE SPECIFICATION>
```

Figure 7.7: Device Specification

7.3 Experimental Evaluation

To validate Decima we divided our experimental evaluation two parts: Evaluation using a real TI setup, and using large-scale PlanetLab experiments.

7.3.1 3D TI System Evaluation

Experimental Setup

As part of our validation, we evaluated Decima in a two-site 3D Teleimmersion System. The first site was located at Siebel Center at the University of Illinois and the second site was located at Research Park at the University of Illinois campus. For visual purposes, we used Bumblebee2 Stereo Cameras as input devices and one display as an output device in each site. In our 3D TI System all the devices were connected to the local CDG through Ethernet.

We have implemented the *Device Class Drivers* for the video and audio devices used in the system. We have also implemented *Physical Device Drivers* as a shim layer on top of the manufacturer drivers for the cameras, microphones, speakers, and displays. We used Google Protocol Buffers [101] to allow language-neutral and platform-neutral data serialization in the implementation of the *Device Control Protocol*. The experiments were all performed on Dell Precision 690 with dual Intel Xeon processor and 2GB of RAM running Linux 2.6.20.

Performance Metrics

We evaluate the performance of Decima in terms of component and end-to-end delay overheads incurred on 1. sending a message from an activity to a device in the local site and 2. sending a *Device Command* from an activity to a device in a remote site.The performance metrics used are defined as follows:

- **Local Site Message Component Delay:** We measure the time required by each component of Decima to deliver a *Device Command* from an activity to a device in the local site.
- **Remote Site Message Component Delay:** We measure the time required by each component of Decima to deliver a *Device Command* from an activity to a device in a remote site.
- **Local Site Message End-to-End Delay:** We measure the total delay incurred by Decima to deliver a *Device Command* from an activity to a device in a remote site.
- **Remote Site Message End-to-End Delay:** We measure the total delay incurred by Decima to deliver a *Device Command* from an activity to a device in a remote site.
- **Call Setup Time:** We measure the total time incurred by an activity in Decima to connect to a device in a remote site.
- **Average CPU Usage:** We measure the CPU usage of Decima and Prometheus at the CDG during steady streaming.
- **Average Streaming End-End Delay:** We measure the end-to-end delay incurred by Decima and Prometheus for streaming.

Results

Local Site Message Delay: We measure the time required by each component of Decima to deliver a *Device Command* (i.e., *startCamera*) from an activity to a device in the local site. We use a control interface implemented as part of an activity (i.e. physiotherapy) to a Bumblebee Camera. Both the control interface and the camera are located in the local site. We measure the time it takes for the delivery and process of the Device Command at the Distributed Device Control Layer (DDCL) and the Device Interface Layer. Component Delay at the DDCL is further divided in: 1. process and delivery of the Device Command from activity to CDG, 2. process and delivery of the Device Command from CDG to device . Figure 7.8 shows that the component delay at the DIL does not exceed 60 μs. Also, Figure 7.8 shows the component delay of the DDCL does not exceed 1500 μs. This shows the low overhead of both the DIL

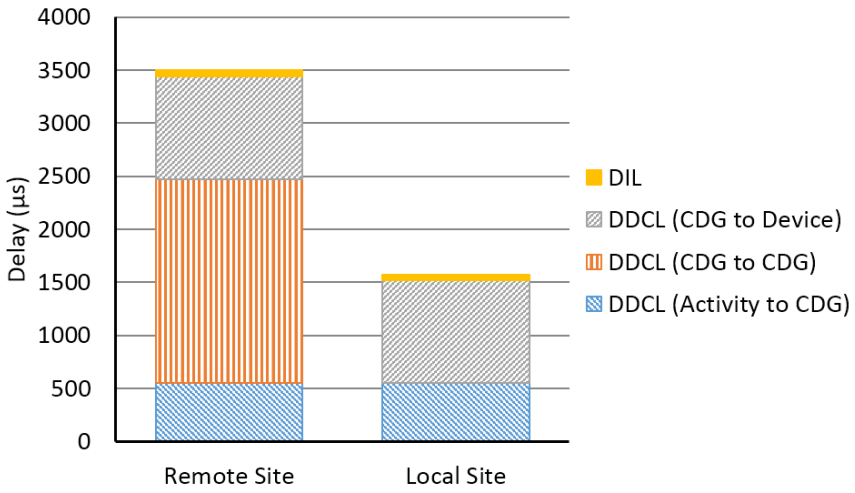


Figure 7.8: Message Component Delay

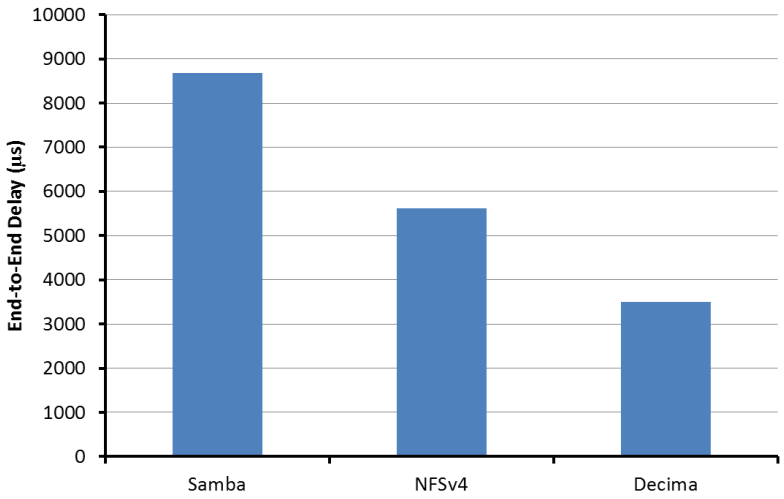


Figure 7.9: End-to-End Delay comparison

and the DDCL virtual layer components. The Local Site Message End-to-End Delay is shown as the sum of all component delays.

Remote Site Message Delay: Similarly, we measure the time required by each component of Decima to deliver a *Device Command* from an activity to a device in a remote site. Component Delay at the DDCL is further divided in: 1. process and delivery of the Device Command from activity to local CDG, 2. process and delivery of the Device Command from local CDG to remote CDG, 3. process and delivery of the Device Command from remote CDG to device at the remote site. Also, this Figure shows the component delay of the DDCL does not exceed 2000 μs. This is a larger value when compared with the local site component delay as the DDCL requires to query the Device Controller to obtain the address of the remote CDG. Remote Site Message EED is shown in

Figure 7.8 as the sum of all the component delays. The end-to-end delay of a `startCamera` Device Command does not exceed 3500 μs. Figure 7.11 compares Decima's Remote EED with the time it takes Samba (SMB) and NFS v4 to perform a file operation in the remote site.

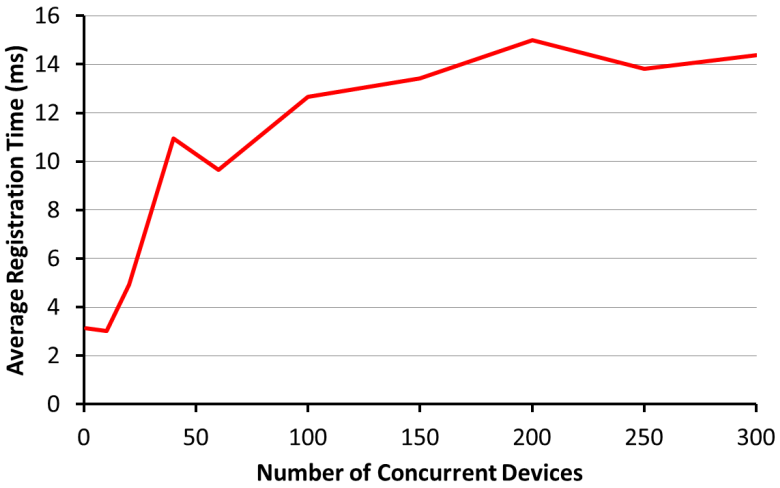


Figure 7.10: Average Device registration time

	Decima and Prometheus	**Yang et al.**	**Device to Device**
Call Setup Time	796 ms	785 ms	765 ms
Avg. Streaming CPU Usage	26.5%	2.7%	Device Specific
Avg. Streaming EED	37.2 ms	35.3 ms	15.3 ms
Interfacing Code	88 XML lines	863 C++ lines	Hardcoded
Multi-Stream Support	Audio and Video	Video only	Video only
Multi-Site Support	Yes	Yes	No
Device Control Support	Yes	No	No

Table 7.2: Comparison of Decima with other systems

Call Setup Time: We measure the total time incurred by an activity in Decima to connect to a Bumblebee Stereo camera in a remote site. This time includes the Device Registration in Decima and the Stream Initialization Time in Prometheus. We compare this value with the call setup time of the system proposed by Yang et al. [7] and with the call setup time between two directly connected devices, i.e., between a directly connected camera and a rendering device without any middleware (i.e., Device-to-Device). Table 7.2 shows the values for the 3 systems.

Average CPU Usage: We measure the CPU usage of Decima and Prometheus at the CDG during steady streaming between two sites using one Bumblebee Stereo Camera to one rendering device in the remote site. We compared this

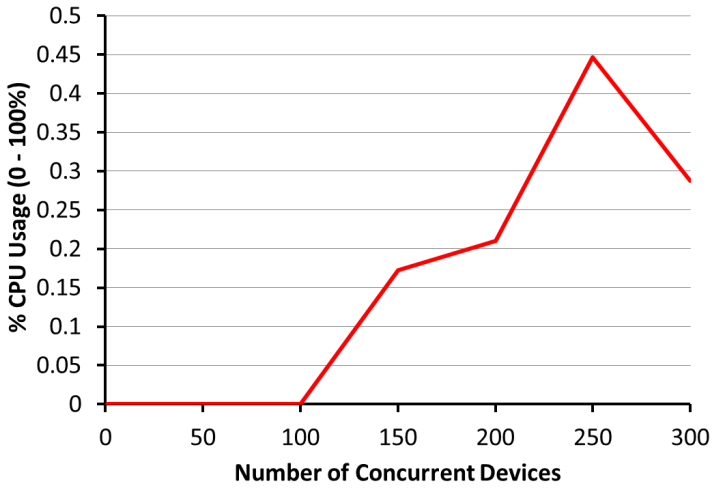


Figure 7.11: DDCS CPU usage

value with the average CPU usage of Yang et al. [7] and with the average CPU usage of two directly connected devices, i.e., between a camera and a rendering device with no middleware (i.e., Device-to-Device). Table 7.2 shows the values for the 3 systems.

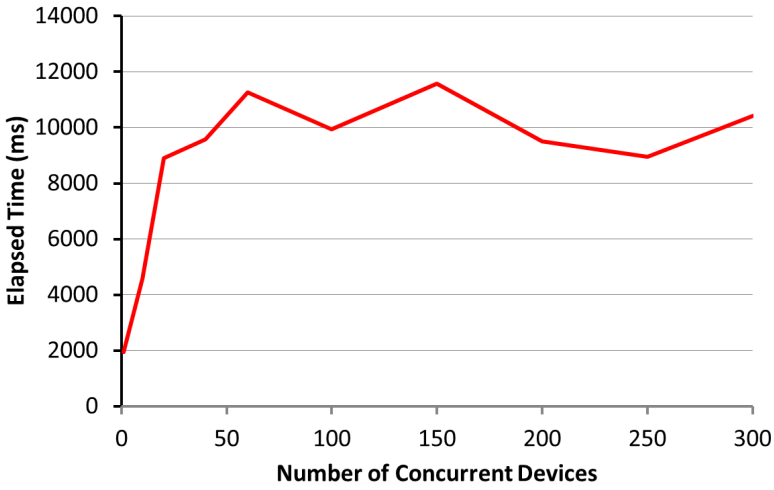


Figure 7.12: DDCS execution time

Average Streaming End-End Delay: We measure the End-to-End Delay (EED) incurred by Decima and Prometheus for streaming between two sites using one Bumblebee Stereo Camera streaming video to one rendering device in the remote site. We compare this value with the average EED of Yang et al. [7] and with the average EED between a camera directly connected to a rendering device with no middleware (i.e., Device-to-Device). Table 7.2 shows the values for the 3 systems.

Comparison with Existing Systems: As part of our validation, we present a comparison of our TI System using Decima with existing TI Systems. Table 7.2 shows the comparison of Decima, the system proposed by Yang et al. [7] and Device-to-Device using no additional virtual layer. The features compared are: average CPU usage while streaming, if the system supports multiple streams (i.e., Multi-Stream), if the system support multiple sites (i.e., Multi-Site) or if the system supports control of devices (i.e., Device-Control). Also, we compare the number of additional lines of code the developer must write as interface to the middleware system for streaming from the device (i.e., Interfacing Code).

7.3.2 PlanetLab Evaluation

Experimental Setup

As part of our validation, we deployed Decima on the PlanetLab. The reason for diverging from the TI testbed is to validate the scalability of the DDCS service with large number of devices. We used one node as the Dynamic Device Configuration Service (DDCS) and simulated TCP/IP camera devices in other PlanetLab nodes. To validate the scalability, we varied the number of concurrent devices registered by a single DDCS node. The setup is shown in Figure 7.13, where nodes "DEV" are the PlanetLab nodes containing simulated Physical Device Driver and Device Class Driver components. Since our goal is to evaluate the scalability of DDCS in this section, we simulate a single device connectivity at each PlanetLab node.

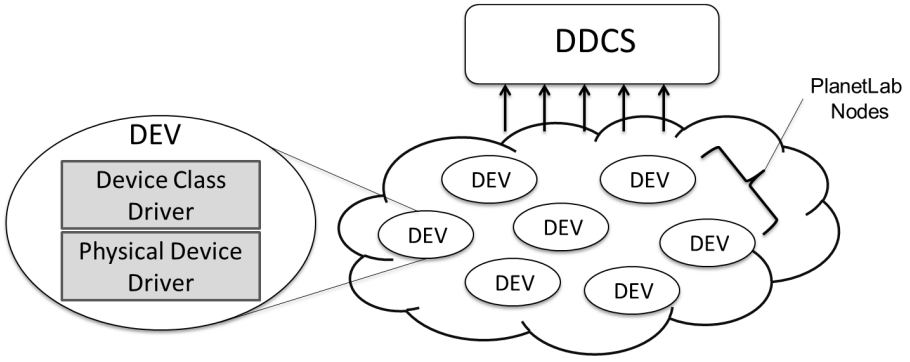


Figure 7.13: Experimental Setup using PlanetLab

Performance Metrics

We evaluate the performance of Decima in terms of overheads incurred on 1. Device Registration Time, 2. DDCS CPU Usage, and 3. DDCS Execution Time . The performance metrics are defined as follows:

- **Device Registration Time:** We measure the total time required by a device to send a Device Configuration Request and receive a HUDI from

the DDCS. From this time we subtract the delay incurred by the network in order to evaluate the scalability of the DDCS independently from the network overhead.

- **DDCS CPU Usage:** We measure the total CPU Usage of the DDCS while processing the concurrent Device Configuration Requests.
- **DDCS Execution Time:** We measure the total time elapsed to process all the device registration request as the number of concurrent devices (PlanetLab nodes) varies.

Results

Device Registration Time: We measure the average time it takes for the DDCS to register a device on different load condition. We subtract the delay incurred by the network in order to evaluate the scalability of the DDCS independently from the network overhead. The load is varied by the number of concurrent devices from 1 to 300. Figure 7.10 shows that the registration time does not exceed 14 milliseconds for even 300 concurrent devices. This emphasizes the low performance overhead of the registration protocol and shows that Decima is suitable for the dynamic and highly contingent applications found in TIs in which devices are dynamically plugged and unplugged frequently.

DDCS CPU Overhead: We measure the average CPU usage of the DDCS while processing the concurrent Device Configuration Requests. We again vary the number of concurrent devices from 1 to 300. Figure 7.11 shows that the average CPU usage is very low at all times and does not exceed 0.5% even for 300 concurrent registration processes. This low CPU usage shows the low overhead and high scalability of the DDCS and shows that Decima is suitable for TI applications where the number of devices connected is large.

DDCS Execution Time: Finally, we measure the total execution time it takes for the DDCS to register all devices for various number of concurrent devices. As Figure 7.12 shows, the execution time does not exceed 12 seconds to process all the concurrent registration requests. Note that the execution time also includes the network delay for requests to arrive which varies up to 200 milliseconds in our PlanetLab setup.

7.4 Conclusion

We introduce device and resource naming protocols based on a Hierarchical Uniform Device Identifier for 3D TI Systems that addresses the challenge of preserving the location-context information of devices and streams with respect of which room and session they belong to. Our hierarchical uniform naming solves the problem of location-context preservation in 3D TI and Telepresence Systems as it allows to uniquely associate a device with a site and a session.

We enable automation of I/O management in TI activities through novel dual virtualization architecture. Our solution, enables seamless dynamism of devices to address the challenge of heterogeneous non-standard I/O devices, i.e., hides against changes in hardware interfaces in multimodal devices (e.g., if the 3D camera used changes from Bumblebee to Kinect). Furthermore, our solution, allows seamless dynamism of activities to address the challenge of contingency and diversity of activities in 3D TI Systems, i.e., enables universal interface of activities to distributed I/O, without any concerns of the underlying software changes (e.g., if user activity changes from walking to sitting).

We validate our architecture with **Decima**, a novel, highly scalable, holistic and context-aware I/O management subsystem for 3D Teleimmersive Systems. Evaluation of Decima with a real TI setup and large-scale PlanetLab testbed shows minimal performance overhead in term of component delay, CPU usage and message overhead.

Chapter 8

Kratos: Activity Management and Detection

3D TI systems are characterized by highly interactive activities that impose tight QoS constraints in terms of bandwidth, delay, skew and jitter. Moreover, activities play a center role in the device and resource management in 3D TI Systems as activities drive the type and number of devices in these systems. For example, simple activities like 3D conversation require only a single camera and a microphone while other more complicated physical activities, like exergaming, might require multiple cameras to cover a much wider field of view required in these activities. Also, Telemedicine activities might require additional body sensors to capture fine grained movement or heart rate variations in the patients. Moreover, different activities define different QoS parameters [14]. For example, 3D Conferencing requires high quality audio and low skew to achieve lip synchronization, whereas exergaming activities like virtual fencing [9] require high frame rate cameras and low delay to achieve high level of interactivity. Finally, processing of streams at each CDG is dependent on the type of activity at the 3D TI System. For example, Multiplayer Online Gaming might require authentication and object collision detection, while a Physiotherapy session might require encryption to ensure doctor-patient confidentiality. As 3D TI Systems are highly driven by activities, a Distributed Operating System for 3D TI Systems must consider activities as a centric part in the resource management of 3D TI Systems. Therefore, activity detection is an important component in Stream OS.

Current approaches to detect activities based on application media data, like the one proposed by Sung et al. [89], require specific knowledge of application data format and therefore, they are unsuitable for 3D TI systems as the diversity of Non-Standard I/O devices and the lack of standard streaming formats make these approaches impractical. Moreover, the interactive nature of activities in 3D TI Systems make these approaches unsuitable as they require computationally expensive image analysis. Niu et al. [88] use several linear motion sensors and aggregate the obtained sensory information to detect 3-D motion patterns associated with specific activities. The problem of this approach is that the space must be augmented with additional sensors in specific locations which might not be available in all 3D TI scenarios.

We propose Kratos, a supervised learning-based *Activity Classification Sys-*

tem that considers application generated metadata and related system metadata (*application-system* metadata) instead of application media data. A supervised learning algorithm trains our classification model for 3DTI setup without the knowledge of media data format or coding complexity. To our knowledge, this is the very first attempt to use time-series application-system metadata in 3D TI activity detection.

Since we classify human activity based on time-series metadata, the classification process we propose here is fast (less than 4ms) and unobtrusive. However, the classification is influenced by several cyber-physical dimensions such as visual color and space volume of the physical contents (e.g., participants) which have impact on the application and system metadata (e.g., the reconstruction time and frame size). We quantify the extent of cyber-physical impacts on our activity classification model, and apply the experimental inferences to classify human activity in a real 3DTI setup efficiently. Using a real 3DTI setup, our solution achieves more than 97% accuracy in human activity classification.

8.1 System Model

8.1.1 Application-System Metadata Model

At each participating site, we monitor application-system metadata information corresponding to application I/O devices and underlying system resources. Application metadata includes camera frame rate, audio bit rate, 3D reconstruction time, rendering time, audio and video frame size. The system metadata includes CPU usage, memory usage and processing time of the participating hosts (such as camera node and gateway node). Every $50ms$, each host generates a snapshot set of metadata values. The combined set of application and system metadata will be referred to as metadata. Metadata values are stored in a timestamp indexed database and are used for activity classification.

8.1.2 Activity Model

We attempt to achieve classification for fine-grained human activities including sitting, standing still and different walk movements performed by the 3DTI participants. We classify fine-grained human activities independently at each site, however, our algorithm can be easily used for multiple sites by running activity detection at each individual site. We also analyze the impact of cyber-physical dimensions on the classification accuracy. As we mentioned before, this classification model can be extended to classify coarse-grained 3DTI activities by combining the contingency and extent of fine-grained activities.

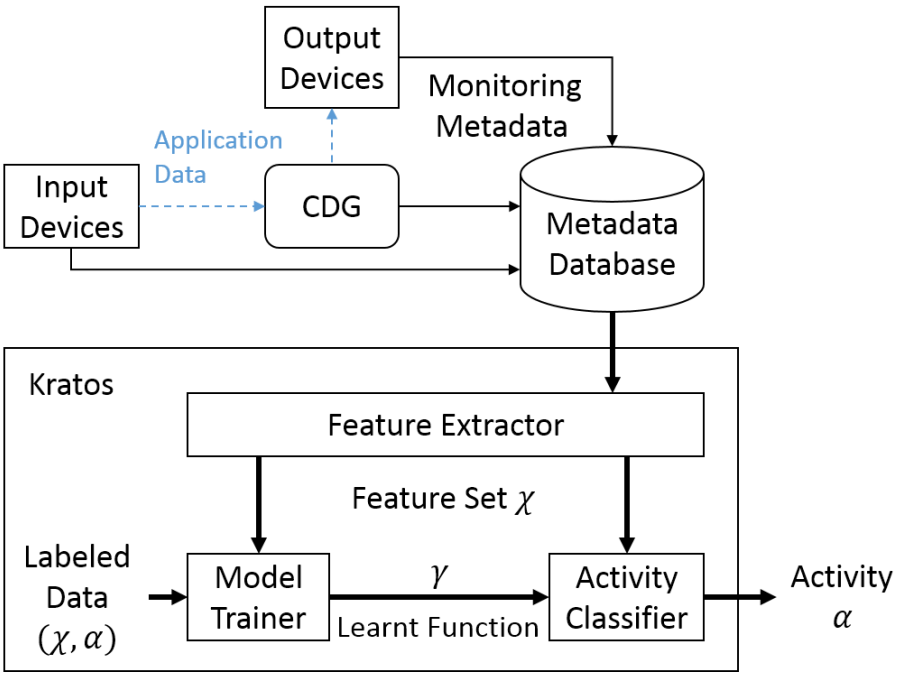


Figure 8.1: Architecture of Kratos.

8.2 System Architecture

The system architecture of Kratos is shown in Figure 8.1. It uses a supervised machine learning approach to classify human activity using application-system metadata. The time-series metadata values are stored in a local database attached to each site. Kratos contains 3 components:

1. *Feature Extractor* that reads metadata from the database and extracts features to construct feature vectors interpretable by a machine learning model.

2. *Model Trainer* that uses these feature vectors and input activity labels to train a classification model.

3. *Activity Classifier* that uses both the feature vectors and the model feeds to finally classify real-time human activity during 3DTI session run-time.

8.2.1 Feature Extraction

Feature extraction transforms recorded application and system metadata into a reduced representation of a set of features (also known as a feature vector). A feature vector is the smallest unit of training and testing. It consists of metadata corresponding to $\mathcal{T}$ consecutive time units. We consider three kinds of features, extracted from the time-series metadata: 1. *Absolute value*, 2. *Time difference value*, which computes the variation of metadata value with respect to the previous time epoch, and 3. *Device difference value*, which computes the

variation of the metadata value generated from the correlated input devices (e.g., all local cameras) at the same time epoch. If there are N metadata parameters $(X_1, \cdots, X_N)$ and x_i^t is the i^{th} metadata value collected at time t, the feature vector (χ) contains: $\cup_{t=1}^{\mathcal{T}} x_i^t$, $\forall_{1 \leq i \leq N}$ (absolute values), $\cup_{t=2}^{\mathcal{T}} x_i^t - x_i^{t-1}$, $\forall_{1 \leq i \leq N}$ (time difference values), and $\cup_{t=1}^{\mathcal{T}} (x_i^t - x_j^t)$ (device difference values), where i^{th} and j^{th} metadata values are generated from correlated devices of the same type (e.g., stereo cameras) and $1 \leq i, j \leq N$.

8.2.2 Model Training

Our objective is to train a classifier using a *supervised machine-learning model* to classify human activity based on 3DTI metadata. To this effect, we use the implementation of SVM [102] by Learning Based Java (LBJ) [103], which is a special purpose framework based on Java for machine learning. Given m labeled samples, containing feature vectors and associated labels $\{(\chi_1, \alpha_1), (\chi_2, \alpha_2), \ldots, (\chi_m, \alpha_m)\}$, where α_i is the class label for the sample feature vector χ_i, the SVM algorithm learns $\gamma : \chi \rightarrow \{-1, +1\}$, where the function γ maps samples χ to a class $\alpha \in \{+1, -1\}$ and is represented by $\text{sgn}(w^{\mathcal{T}} \chi - \Theta)$, where $w \in R^n$ is the weight vector and $\Theta \in R$ denotes the constant threshold.

Since we consider multiple activities, for multi class classification, the LBJ library implementation of SVM employs the one to all strategy. This strategy learns independent binary classifiers for each class label α_i treating a sample (χ_i, α_i) as a positive training sample $(+1)$ only for class label α_i and negative (-1) for all other class labels.

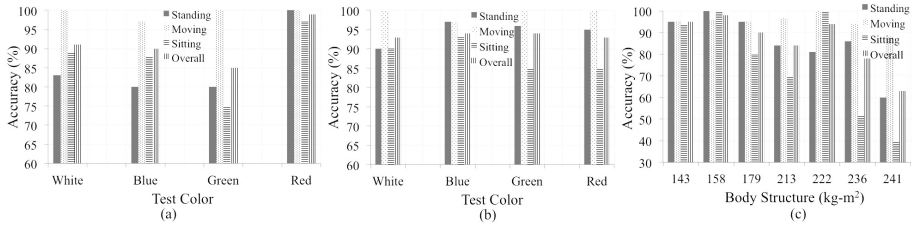


Figure 8.2: Activity classification accuracy with training model for (a) SC=red (Phase 1), (b) SC values not included in the test (Phase 2), and c) participant having BdS=176 (Phase 1).

8.2.3 Activity Classification

Once the model has been trained, it predicts the human activity given a feature vector. After learning independent classifiers (weight vectors w_j) for each class label α_j, the model performs the following calculation: $\gamma(\chi)$=argmax$_j$ $w_j^{\mathcal{T}}(\chi)$, where χ is the to-be-classified feature vector, w_j is the weight vector learnt for the j^{th} activity class and $\gamma(\chi)$ is the predicted class label.

8.3 Experiments

As part of our validation of Kratos, we perform the following experiments:1. Study the impact of certain cyber physical parameters on our activity classification model, and 2. Employ an efficiently trained model to classify activities.

8.3.1 Effect of Cyber Physical Parameters

We realize that human activity classification accuracy is influenced by certain cyber-physical dimensions. We consider two cyber-physical dimensions in our current study: 1. Visual color of the cyber-physical content, and 2. Volume of the cyber-physical content. Since the background is usually subtracted, the visual color and physical texture of human body are the main image features that impact frame size metadata in mesh-based 3D reconstruction of foreground image. Likewise, the volume of the foreground content impacts frame size value of the constructed image. Change in frame size metadata changes 3D reconstruction time, data transmission time, network bandwidth and hence frame rate.

The visual color dimension is dependent upon physical lighting condition, shirt color as well as skin color of the participants. The volume dimension is dependent upon the participants' weights and heights. To quantify the impact of these two cyber-physical dimensions, we consider two metrics: 1. *Shirt Color (SC)*, and 2. *Body Structure (BdS)* . We consider four different shirt colors: white, blue, green, and red. The Body Structure is based on the Body Mass Index (BMI) [104], we define the BdS as follows: $BdS = H^2 \times W$, where H denotes the height of the participant in meters and W denotes the weight of the participant in kilograms.

Experimental Setup

For our validation, we use a 3DTI site with one gateway, two (upper and lower body) 3D Bumblebee2 cameras (constructing mesh based 3D video streams via connected computers), one renderer and one database node. We run Kratos on the local database machine and attempt to classify between a basic activity set $A = \{stand, sit, walk\}$. For each activity session, 4 minutes of metadata is recorded. The data is collected under constant $7200K$ white lighting conditions. People involved in the experiments are given basic instructions on how to perform the respective activities, without any further information on how the application would classify their activity. Each classification unit is constructed with $\mathcal{T} = 50$ and $N = 20$, which are the customizable parameters in 8.2.1.

To understand the impact of cyber-physical dimensions on human activity classification, we run experiments in two phases. In **Phase 1**, we train the model on one random cyber-physical dimension value from the dimension spectrum (e.g., SC=blue) and record the classification accuracies given by the model on

testing with other values (e.g., SC=red). In **Phase 2**, we train the model on a more representative set (i.e., with multiple values) from the cyber-physical dimension spectrum, and record the classification accuracies on a random value (not included in the training).

Results and Discussion

Impact of Shirt Color (SC). To understand the impact of the Shirt Color (SC) dimension, we minimize the impact of the Body Structure by recording application-system metadata for the same person under the same white light condition. We show the result for Phase 1 in Figure 8.2(a). A high level of accuracy ($> 85\%$ overall) is achieved for activity classification, where training is done based on activity (i.e., moving, standing and sitting) metadata collected only for SC=red. Changing the Shirt Color value of the participants shows limited impact on classification accuracy. We can conclude that the skin color of the participants and the lighting condition will not have high impact on activity classification accuracy either, since the learning is done based on the time-series patterns (which are similar across different Shirt Color values) of metadata.

In Phase 2, the model is trained on data corresponding to all Shirt Color values except the Shirt Color value included in the test. The model achieves an average accuracy of about 94% as shown in Figure 8.2(b). This implies that when the ML algorithm is exposed to data corresponding to a wide range of Shirt Color values during training, the impact that the change in color has on the features helps it achieve higher accuracies for other values in the spectrum (even though the particular Shirt Color value is not seen during training).

ID	**Height (m)**	**Weight (kgs)**	**Body Structure** ($kg \cdot m^2$)
1	1.73	59.0	176
2	1.70	54.5	158
3	1.80	65.7	213
4	1.75	72.5	222
5	1.77	77.0	241
6	1.66	65.0	179
7	1.62	54.5	143
8	1.77	75.0	236

Table 8.1: User data of body structure.

Impact of Body Structure (BdS). To study the effect of body structure, activity data is collected for 8 different people, all wearing a red shirt to minimize the impact of Shirt Color dimension. Table 8.1 shows the collected Body Structure data for experiments' participants. The Body Structure values range from 143 to 241.

In Phase 1, where the training of Body Structure metadata is taken from the lower end of the spectrum ($BdS = 176$), we observe a general trend of dropping

classification accuracies as we walk towards the higher end of the spectrum ($BdS = 236$), with the overall (for all fine-grained activities) accuracy dropping as low as 65% (Figure 8.2(c)).

This is because different cameras (upper and lower body cameras) get different spatial coverage of the participants and therefore the differences in the metadata between the training and the testing phases is significantly different. This implies that Body Structure shows high impact on the activity classification. This is especially true for the sitting activity, where the upper body and lower body camera captures different volume dimensions if we abruptly change the Body Structure (BdS) value from the training to the testing phase. In this case, the classification accuracy falls to about 40%.

However, in Phase 2, our classification model achieves an overall average accuracy of 97% whereas the training set included activity data corresponding to all Body Structure values except the test Body Structure value. This implies that if the machine learning model is trained with a representative set of data distributed uniformly over the whole Body Structure spectrum, it is able to accurately classify activities for any data points throughout the spectrum.

8.3.2 Activity Classification

Experimental Setup

The experimental setup was the same as 8.3.1, with the exception that the model is trained to classify between a basic activity set A = {stand, sit, walk00, walk01, walk10, walk11}, where the definitions of the walking activities (walkXY) are given in Table 8.2.

	Coverage = Low	**Coverage = High**
Speed = Low	walk00	walk01
Speed = High	walk10	walk11

Table 8.2: Walking Activity Definition

The $Coverage \in \{Low, High\}$ refers to the spatial coverage of the participant in the physical space and the $Speed \in \{Low, High\}$ refers to the speed of movement and limb actions [92]. For each activity listed, 4 minutes of metadata was recorded for 5 people across the Body Structure spectrum wearing the same color shirt (SC=RED), since earlier experiments in 8.3.1 have shown that body structure has a significant impact on activity classification accuracy.

Results and Discussion

The trained model is tested on random activity sessions' metadata, that is not seen during training. Table 8.3 the percentage of activity samples observed and the corresponding resulting activity classification.

Observed Activity	Resulting Classified Activity					
	walk00	*walk01*	*walk10*	*walk11*	*stand*	*sit*
walk00	47.50	0.00	32.50	0.00	17.50	2.50
walk01	2.17	78.26	8.70	8.70	2.17	0.00
walk10	7.14	0.00	88.10	4.76	0.00	0.00
walk11	2.56	38.46	5.13	53.85	0.00	0.00
stand	0.00	0.00	4.17	0.00	95.83	0.00
sit	0.00	0.00	0.00	3.70	0.00	96.30

Table 8.3: Activity Classification Distribution (%)

Based on the results, the following observations are made: 1. Our model performs very well in classifying Stand and Sit. 2. Classification between different movements is harder due to the minor variation in the impact of the different types of movement on metadata. However, we can still achieve 88% accuracy for walk01 and 78% accuracy for walk10. 3. The major mis-classification occurs due to the variation in *Speed*; walk00 is frequently mis-classified as walk10, and walk11 is frequently mis-classified as walk01. 4. In all cases, the *Coverage* dimension is classified correctly.

8.4 Conclusion

We propose Kratos, a system that addresses the activity classification problem in 3DTI systems from an unique perspective, using time-series application-system metadata in 3D TI Systems. Our activity classification system addresses the challenges posed by multimodal interfaces by using application-system metadata as opposed to application media data. Our approach using an SVM is decoupled enough that can be extnded to other activities and therefore it is suitable for systems where activities are very diverse. With a few exceptions, our system is able to achieve high accuracy for classifying a basic set of human activities in a running 3DTI session. The detection process is fast (4 ms) and therefore is suitable for fast-paced interactive activities as those found in 3D TI systems. Moreover, the system provides a base for constructing a classification system for coarse-grained activity detection.

Chapter 9

Hera: Offline Resource Profiler

3D TI systems are characterized by highly interactive activities that impose tight QoS constraints in terms of bandwidth, delay, skew and jitter at the local Content Delivery Gateway (CDG) in each site. Moreover, activities play a center role in determining the QoS requirements of a 3D TI CDG in several ways:

1. The nature of the activity drives the *type and number of devices* used in the 3D TI System. For example, simple activities like 3D conversation require only a single camera and a microphone while other more complicated physical activities like exergaming might require multiple cameras to cover a much wider field of view required in these activities.

2. Different activities define *different QoS parameters*. For example, 3D Conferencing requires high quality audio and low skew to achieve lip synchronization, whereas exergaming activities like virtual fencing [9] require high frame rate cameras and low delay to achieve high level of interactivity.

3. Frames from 3D video cameras have *different rate and bandwidth* based on the complexity of the captured scene, as complex scenes require more processing time and a higher number of pixels to be represented. For example, the average frame size of a 3D video frame with only one person standing is 11 Kb, however the average frame size of a 3D video frame with on person sitting is 5 Kb.

4. *Processing of streams* at each CDG is dependent on the type of activity at the 3D TI System. For example, Multiplayer Online Gaming might require authentication and object collision detection, while a Physiotherapy session might require encryption to ensure doctor-patient confidentiality.

This variability of QoS requirements for each activity in 3D TI, creates the problem of determining the QoS parameters at the Content Delivery Gateway (CDG) for each session based on the type activity, the devices involved in the 3D TI session and the processing functions that will be applied to each stream. One approach to deliver QoS requirements is over-provisioning of resources. However, the high bandwidth requirement and the large number of streaming devices make over-provisioning a prohibitive approach. Therefore, we will use admission control for usage of StreamOS resources. However, to enable efficient admission

control, StreamOS requires precise specification of the QoS requirements of each session. To solve this problem, we propose Hera, an offline profiling tool that allows users to determine the QoS requirements, in terms of CPU and bandwidth of a 3D TI session. Hera provides estimation of QoS parameters based on previous statistical data and reduces the amount of over-provisioned resources that must be reserved to meet the QoS requirements of the 3D TI session.

Real-Time Schedulers with adaptation mechanisms like GraceOS [47], DSRT [23], and RK [45] address the variability of the QoS requirements by incrementing the Computation Time assigned to a particular Task from an overrun budget based on the overall laxity of the system. Unfortunately, this approach still requires prior information about the QoS requirements of the application and can only provide limited elasticity of the resource demand.

Other existing approaches, like iDSRT [46] provide online resource profiling, however, these Real-Time schedulers do not explicitly consider activity as a specific constrain and only rely on initial run-time information. This approach can be misleading as users in 3D TI might take significant time before they engage in some specific activity while the 3D TI System is running. Also, these schedulers do not consider the time and space correlation between streams in 3D TI systems, neither do they consider that different activities require different processing and therefore they are unsuitable for 3D TI Systems.

We propose a QoS model for 3D TI Systems that enables estimation of the QoS requirements in terms of CPU and bandwidth of a 3D TI session based on *offline statistical information*. Our QoS model considers the Visual Space of a 3D TI Session to address the variation in the QoS requirements due to the changes in the processing time of a frame due to its complexity. Our model also considers the Activity to address the variability in the QoS requirements for each activity. As part of our model, we also consider activity-driven processing functions, as each activity requires different processing functions and therefore, the Computation Time requirements differ for each activity. We use this model to design Hera, an offline profiling tool, that provides estimation of QoS parameters based on previous statistical data and reduces the amount of over-provisioned resources that must be reserved to meet the QoS requirements of the 3D TI session.

9.1 Hera Profiling Architecture

Hera interfaces with the hardware and software implementing the 3D TI System and uses an online monitor [105] to collect system metadata and profile the QoS requirements of a 3D TI session. It is composed of 3 main entities: 1. *Online Monitor*, 2. *Profile Database* and 3. *System Analyzer*.

Initially, the system is trained by running the 3D TI System for various activities. During the *Training Phase*, the Online Monitor is responsible for capturing metadata information about the data streams produced by each device

in the 3D TI System. This metadata includes timestamps and sizes for each frame of each device along with CPU utilization for each of the processing functions at the Content Delivery Gateway (CDG) in the local 3D TI site. This metadata information is indexed and stored in the Profile Database for use during the *Profiling Phase*.

After the initial Training Phase, Kratos (Chapter 8) is responsible for detecting the type of the activity in the 3D TI system and notifies Hera. The System Analyzer of Hera determines the QoS parameters of the 3D TI session based on the activity in the 3D TI System. The Session Description, obtained by aggregating information aggregated from each component of StreamOS, contains the information about the type of activity and the type of devices that are in use during the 3D TI session. The System Analyzer uses the Session Description along with the information from the Profile Database to determine the Session QoS Parameters. Session QoS Parameters are used by StreamOS to provide estimated soft real-time QoS guarantees for the 3D TI session.

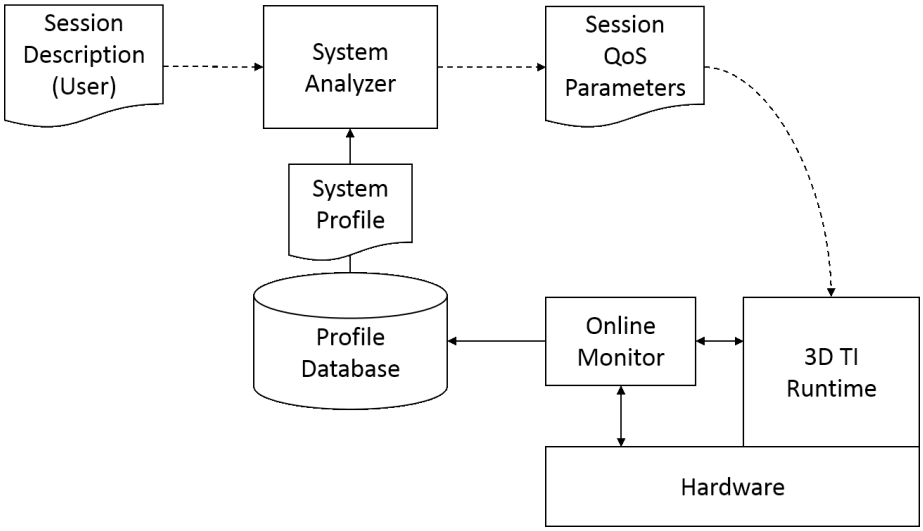


Figure 9.1: Hera Architecture

The System Analyzer uses a QoS model specially tailored for 3D TI Systems. This QoS model is detailed in Section 9.2 and builds on the ideas presented in Chapter 3 and Section 5.2.

9.2 Hera QoS Model

The goal of our model is to capture the Session QoS Parameters that define the CPU and bandwidth requirements of the CDG for a 3D TI session based on the Session Description, including the type of activity, the type of devices used in the 3D TI system and the type of processing functions applied to each stream at the CDG. More formally, we define the Session Description as a 5-tuple: $(a, VS, \{F_i\}_{i=1}^{m}, Dev, q)$, where a is the activity in the 3D TI System, VS is the

Visual Space, Dev is the corresponding set of devices, $\{F_i\}_{i=1}^{m}$ is a sequence of processing functions applied to each stream generated by the devices and q is a quality parameter ($0 < q \leq 1$) that defines the level of desired QoS for the 3D TI session. The Visual Space VS can be defined as the most contributing color in the 3D TI scene. Since background pixels are removed from the 3D TI Streams, the most contributing color is usually the clothing color of the participants. Our previous experience [106] indicates that the Visual Space plays a significant role in the QoS characteristics of the 3D TI Streams as activity detection improves significantly when Visual Space is considered as part of the 3D TI Model.

As mentioned in Section 5.2, at the local Content Delivery Gateway each Device gets mapped into an Instream Process and an Outstream Process. Therefore, our Session QoS Parameters must consider two separate CPU requirements for each device (i.e., one for the Instream and one for the Outstream). Formally, we can define the Session QoS Parameters for a device Dev as a 5-tuple: $(C^{In}, C^{Out}, P_S, BW^{In}, BW^{Out})^{Dev}$, where C^{In} is the number of CPU-time units required by the Instream process each period, C^{Out} is the number of CPU-time units required by the Outstream process each period, P_S is the period of the Stream S, BW^{In} is the Network Bandwidth required to receive the Stream and BW^{Out} is the Network Bandwidth required to send the Stream.

Figure 9.2 shows an overview of the QoS Model as input and output parameters of the System Analyzer.

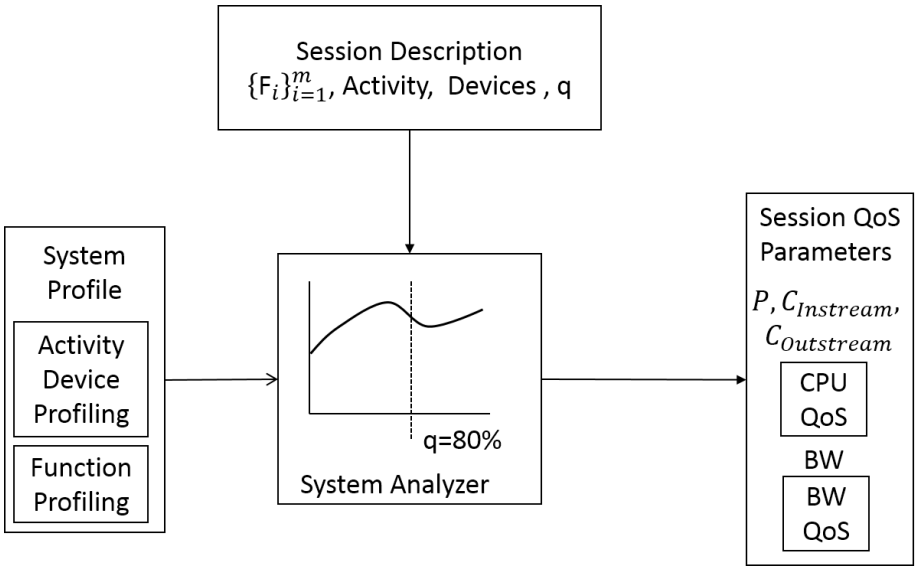
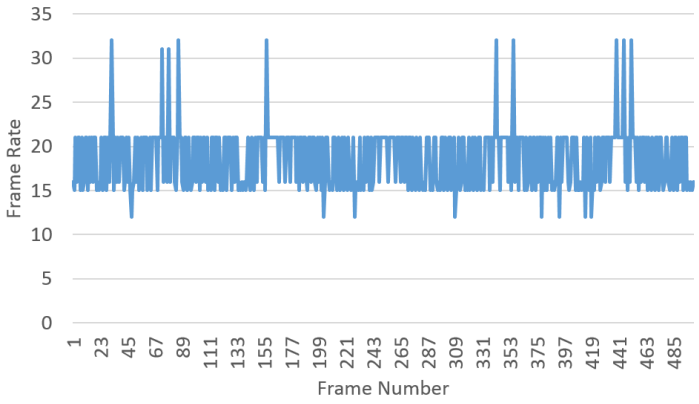


Figure 9.2: Hera System Analyzer Model

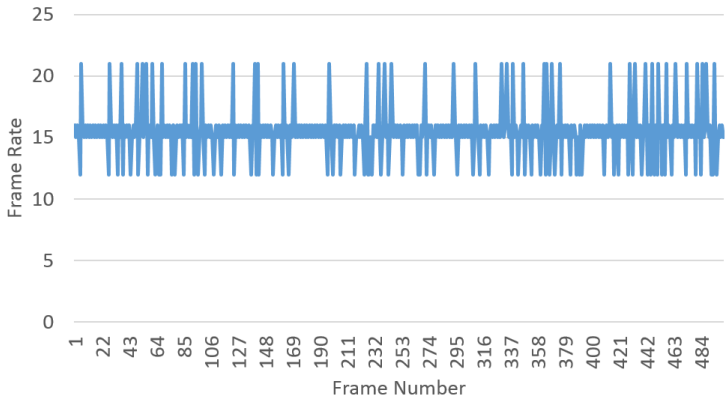
9.2.1 Device Model

3D TI Systems are composed of streaming devices. Frames from 3D video cameras have different rate and bandwidth based on the complexity of the captured

scene, as complex scenes require more processing time and a higher number of pixels to be represented. To model this variability, we define a Variable Rate Stream as an ordered pair: $\tilde{S} = (\{f_k\}_{k=1}^{n}, \{\Delta_k\}_{k=1}^{n})$, where f_k represents the k^{th} frame in a sequence of frames and Δ_k is the time interval between frame f_k and frame f_{k+1}. We formally define such an interval as: $\Delta_k = t_{k+1} - t_k$, where t_k and t_{k+1} are monotonically increasing arrival timestamps for frames f_k and frame f_{k+1}. It is important to note that our model assumes that frames do not arrive out of order as Prometheus uses TCP/IP as the underlying transport protocol.



(a) Frame Rate Trace for Walking Activity



(b) Frame Rate Trace for Standing Activity

Figure 9.3: Trace of the Frame Rate exhibiting weakly periodic behavior

3D TI video Streams are weakly periodic. Figure 9.3 shows a trace of the frame rate of a 3D TI session with walking and standing activities exhibiting such weakly periodic behavior. In order to use Real-Time periodic schedulers, we need to shape the weakly periodic incoming traffic into strong periodic that can be scheduled by Real-time periodic schedulers like EDF. Therefore, the goal of our model is to the best constant rate that approximates a Variable Rate Streams into a Periodic Streams $S = (\{f_k\}_{k=1}^{n}, P)$. Our approximation

uses Gaussian Likehood Density Estimation to statistically model the period and size of a Variable Rate Stream as Normally Distributed.

Given a sequence of time intervals $\{\Delta_k\}_{k=1}^{n}$ between frames of a Variable Rate Stream $\tilde{S} = (\{f_k\}_{k=1}^{n}, \{\Delta_k\}_{k=1}^{n})$; for each frame f_k, we define a Random Variable $X_k(\Delta_k) = \Delta_k$ with sample space $\Omega_k^X = \{x | x > 0\}$.

Let $\bar{X_k}$ be the sample mean of X_k for a sufficiently large sample size. Based on the Central Limit Theorem we assume that the distribution of $\bar{X_k}$ for $k < n$ is Normal with Probability Density Function defined as $\bar{X_k} \sim \mathcal{N}(\mu, \sigma^2)$. Then, we approximate $\tilde{S} \approx S = (\{f_k\}_{k=1}^{n}, P_S)$, where $P_S = \frac{1}{\mathrm{CDF}^{-1}_{\bar{X_k}}(q)}$ and q is the probability that $\Delta_k \leq P_S$ $(Pr(\Delta_k \leq P_S) = q)$.

Similarly, we estimate the Normal Probability Density Function for the frame size of stream $\tilde{S}$. Let z_k be the size of frame f_k. We define a Random Variable $Y_k(z_k) = z_k$ with sample space $\Omega_k^Y = \{y | y > 0\}$. We assume that the distribution of $\bar{Y_k}$ for $k < n$ is Normal with Probability Density Function defined as $\bar{Y_k} \sim \mathcal{N}(\mu, \sigma^2)$. Then, we approximate the largest frame size of S as $Z_S = \frac{1}{\mathrm{CDF}^{-1}_{\bar{Y_k}}(q)}$ with probability $Pr(z_k \leq Z_S) = q$.

In order to address the challenge of time and space correlated streaming our model considers the concept of Stream Dominance [93] based on the Visual Contribution Factor as proposed by Yang et al. [107]. A Dominant Stream S^D is define as the stream with the maximum Visual Contribution Factor. In our model we consider separate distributions from Dominant and Non-Dominant streams. Our experimental evaluation shows significant differences in QoS requirements for Dominant and Non-Dominant Streams in activities with large difference in Visual Contributions Factors between Dominant and Non-Dominant streams.

9.2.2 CDG Model

Stream Dissemination

3D TI Content Delivery Gateways are responsible for processing and disseminating 3D TI Streams to remote CDG. Each stream S is mapped into two streams: an Input Stream (Instream) and an Output Stream (Outstream), more formally denoted as $S \rightarrow (S^{In}, S^{Out})$. Therefore, each stream S is mapped into two underlying Tasks in the Traditional Operating System. Computation Time of receiving a frame from an Instream from the network or sending a frame from an Outstream to the network can both be modeled as linear functions of the frame size.

Given an Instream $S^{In} = (\{f_k\}_{k=1}^{n}, P_S)$ where the frame size of frame f_k is z_k, we define a regression for the Computation Time of receiving a frame from an Instream, where the dependent variable is the Computation Time denoted as C_{In}^S and the independent variable is the size of the frame denoted as z. Formally, we define the regression model as $C_{In}^S \approx \varphi_{In}(z, \beta_{In})$, where φ_{In} is the linear regression function and β_{In} represents the parameters of the regression

function. Similarly, we can define a regression model for the Computation Time of sending a frame from an Outstream as $C^S_{Out} \approx \varphi_{Out}(z, \beta_{Out})$.

Stream Processing

As mentioned in Chapter 3, processing of 3D TI Streams is modeled as a Computational Pipeline defining a composition monoid. In our model, a stream, either Instream S^{In} or Outstream S^{Out} is mapped into a Task T_i and each frame f_k of a stream is associated with a Job $J^{T_i}_k$ in which the frame f_k is processed through a sequence of functions $\{F_{ii}\}^m_{ii=1}$.

To model the Computation Time of each processing function we assume that the running time of most algorithms used in 3D TI are proportional to the size of the input frame. Therefore, for each processing function, we can define a generalized regression model in which the dependent variable is the Computation Time denoted as $C^S_{F_{ii}}$ and the independent variable is the size of the frame denoted as z. Formally we can define the regression model as $C^S_{F_{ii}} \approx \varphi_{F_{ii}}(z, \beta_{F_{ii}})$, where $\varphi_{F_{ii}}$ is the function used for the regression and $\beta_{F_{ii}}$ are the parameters of the function.

Many of the algorithms in 3D TI Systems are non-linear, therefore each processing function will have a different regression function. In order to determine the best regression function programatically, Hera computes the regression function using various functions including linear, quadratic and logarithmic and uses the model that yields the smallest Sum of Squares Error (SSE). Algorithm 5 shows this algorithm in detail. Our algorithm takes a set Π as input, containing all the potential regression models (e.g. $\Pi = \{\phi^{constant}, \phi^{linear}\}$). Our algorithm works by computing all the potential regression ϕ in the set Π and computing the Sum of Squared Error for each of them. The algorithm then uses the regression function ϕ with the smallest error.

Algorithm 5 Hera Generalized Regression Model

Input: Set of valid regression models Π, $C^S_{F_{ii}}$ and z
Output: $\varphi_{F_{ii}}$ and $\beta_{F_{ii}}$

```
SSE_min ← ∞
for all φ ∈ Π do
    Compute C^S_{F_ii} ≈ φ(z, β_φ)
    if SSE(C^S_{F_ii}, z, β_φ) < SSE_min then
        SSE_min ← SSE(C^S_{F_ii}, z, β_φ)
        φ_{F_ii} ← φ
        β_{F_ii} ← β_φ
    end if
end for
return φ_{F_ii} and β_{F_ii}
```

Aggregated Computation Time

In the case of an Instream, the sequence of functions is processed after the stream is received from the network. In the case of an Outstream the sequence of functions is processed after the stream is sent to the network. Therefore, the Computation Time required to process and disseminate an Instream or Outstream, is the total Computation Time required by processing all the functions in the sequence $\{F_{ii}\}_{ii=1}^{m}$ plus the time required to receive or send a frame of the stream through the network, respectively denoted as C_{In}^{S} or C_{Out}^{S}

In our model, we assume a worst case time analysis, where the worst case is determined by the QoS parameter q provided by the user in the Session Description. Therefore, to compute the Computation Time, we assume that the frame size is determined by the Device Model as $Z_S = \frac{1}{CDF_{Y_k}^{-1}(q)}$.

Then, we define the Computation Time C_i of an Instream Job for Task T_i with underlying stream S as: $C_i = C_{In}^{S} + \sum_{ii=1}^{m} C_{F_{ii}}^{S}$, where $C_{F_{ii}}^{S}$ represents the Computation Time required for processing function F_{ii} for a frame of size Z_S and C_{In}^{S} represents the Computation Time required to receive a frame of size Z_S from the network. Similarly, we can define the Computation Time C_j for an Outstream Job for Task T_j with underlying stream S as: $C_j = C_{In}^{S} + \sum_{ii=1}^{m} C_{F_{ii}}^{S}$.

Bandwidth Estimation

Our Bandwidth Estimation is based on the average utilization case. Our model only considers application level payload and does not consider the overhead caused by underlying network protocols. Our model considers separate bandwidth allocation for Instreams and for Outstreams. To estimate the bandwidth required to receive a stream, our model uses the following computation $BW^{In} = \frac{Z_S}{P_S}$, where Z_S is the average frame size and P_S is the Period of Stream S. We assume that the bandwidth required for sending or receiving a stream is symmetrical, therefore, the bandwidth required to send Stream S can be defined in a similar manner: $BW^{Out} = \frac{Z_S}{P_S}$.

9.3 Experimental Evaluation

We evaluated Hera in a 3D TI System, with 2 sites. Each 3D TI Site was composed of 2 Bumblebee 3D Stereo Cameras as devices. The Content Delivery Gateways used were Dell Precision 690 with a Quad-core Intel Xeon Processor. As part of our evaluation, we considered 3 activities: sitting, standing and walking and we also considered 3 different Visual Spaces based on 3 different shirt colors worn by the same participant: Red, Green and Blue.

Our validation of Hera is divided in 3 parts:

1. Validation of our Device Model in terms of precision of our estimation when compared to the observed frame rate and frame size, i.e, how close

the model can fit data from a live session. For this experiment we use a cumulative histogram of data captured form a live session and compare the percentage of underestimation or overestimation between our model and the cumulative histogram.

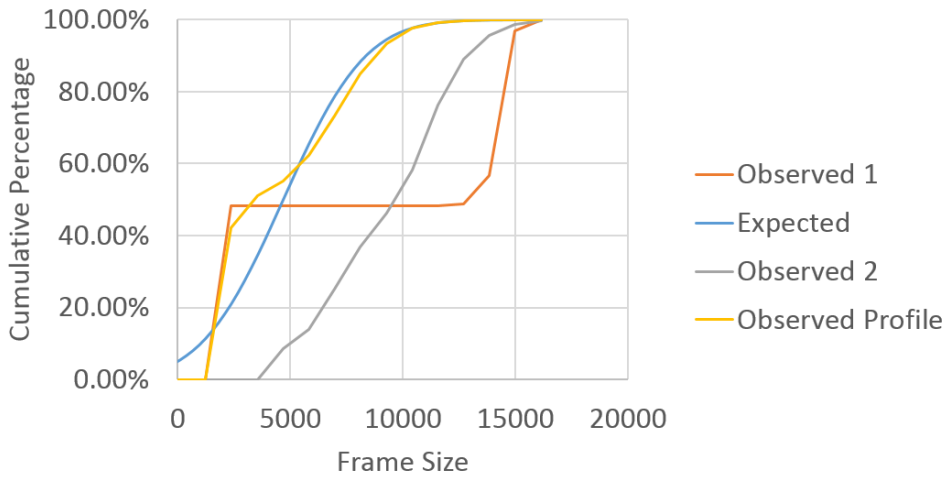
2. Validation of our CDG Model in terms of precision of the regressions when compared to the observed computation time, i.e. how close the regression can fit data from a live session. For this experiment we use the Coefficient of Determination R^2 as our metric. We use the Coefficient of Determination as a measurement of how well the data points fit the particular regression function obtained by our model.

3. Comparison of our Device Model with other approaches in terms of the precision of these estimations when compared to the observed frame rate and frame size.

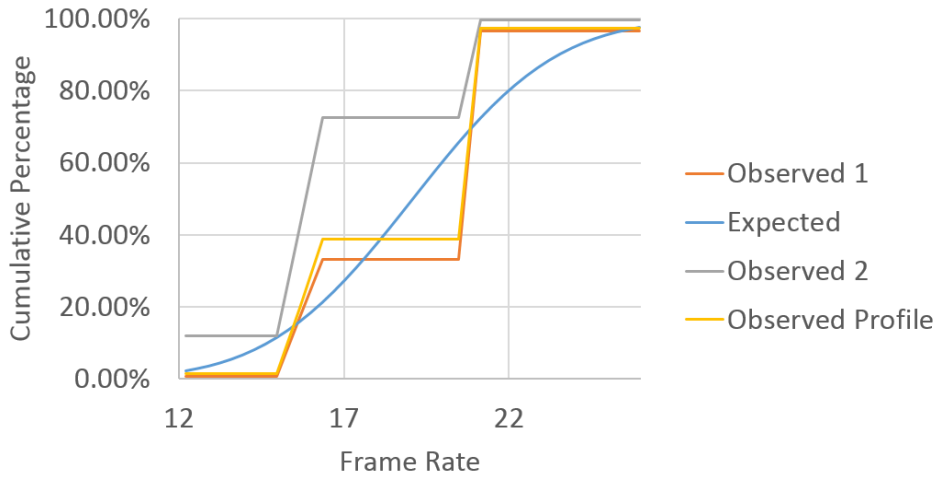
As our first experiment, we validate our Device QoS model by comparing how precise our model can predict the Device QoS requirements of the users in terms of stream period and frame size with the profiled data (i.e., training data). We use Red shirt color as the Visual Space reference (i.e., training data), and we compare the training data with the participant that trained the model for 3 activities: sitting, standing and walking. We compare the Cumulative Distribution Function (CDF) obtained by our model (i.e., Expected) with a histogram of a live session for each activity (i.e., Observed Profile). Figures 9.4, 9.5 and 9.6 show the results of our experiment.

Our validation shows that the quality of the prediction is highly dependent on the type of activity, as some activities are easier to fit as they have much less variability and therefore smaller variance when fitted in the model. In our experiments, we find that Standing and Walking are easier to predict. In the case of Walking, our experimental evaluation shows that our model underestimates frame sizes by at most 5% for values above the mean ($q >= 50\%$). In the case of the Standing activity our experimental evaluation shows that our model overestimates the frame size for values close to the quality parameter $q > 72\%$ by 20% and underestimates the QoS requirements by 25% for values near the mean ($q = 50\%$). Also, our experimental evaluation shows that frame rate also shows smaller variation than the frame size, and therefore, it is easier to fit through a parametric approach, like the one proposed in our model. For the case of the Standing activity, our model underestimates the frame rate by only 2% when the quality parameter is $q > 97\%$, but is underestimated by 25% near the mean ($q = 50\%$).

As our second experiment, we evaluate our Device QoS model by comparing the profiling data (i.e., training data) with 2 previously unseen participants to further measure how precisely our model can predict the stream period and frame size for participants that are not in the training data. We use Red shirt



(a) Frame Size Model Evaluation



(b) Frame Rate Model Evaluation

Figure 9.4: QoS Device Model Evaluation for Walking Activity with Red Visual Space

color as Visual Space and we compare the profiling data with 2 other participants (i.e., Observed 1 and Observed 2). Figures 9.4, 9.5 and 9.6 show the results of our experiment.

A part of our evaluation, we wanted to validate how important is the Visual Space Contribution parameter in our Device QoS model. For this experiment, we have 2 separate training sets: A training set where the person is wearing a Red shirt. 1. A training set where model is trained with the person wearing 3 shirts, each with a different color (i.e., Red, Green and Blue). The activity performed in this experiment was Standing. We use the Standing activity for this experiment as this activity was the best fitted activity in the model. We compare the training data with the same participant that trained the model.

Our evaluation shows that *Visual Space Contribution is very significant* in

our model. Figure 9.7(a) shows that the Red Profile overestimates the QoS frame size by 38% for $q = 99\%$ for the Green Visual Space and overestimates it by 20% for the Blue Visual Space. However, for the case of the frame rate, the Visual Space Contribution is not as significant. Figure 9.7(b) shows that frame rate is only overestimated by 24% for the Green Visual Space and by 21% for the Blue Visual Space.

As part of our validation, we wanted to validate how important is the concept of Activity in our Device QoS model. For this experiment, we have a training set that considers the 3 activities: Walking, Standing and Sitting in the same training set. We compare the training data with the participant that trained the model performing each of the same 3 different activities. For the entire experiment, we used a Red Visual Space. Figure 9.8 shows that the concept of Activity is very important in our Device QoS model. In Figure 9.8a, we can observe that if we do not consider activities, the the Red Profile overestimates the frame sizes for the standing activity by 35% and for the sitting activity by as much as 59% for $q = 75\%$. For the case of the frame rate, Figure 9.8b shows that the profiled data overestimates the frame rate by 30% for the walking activity, it underestimates the frame rate for the standing activity by 26% and it underestimates the frame rate for the sitting activity by 17% for $q = 65\%$. This shows that 3D TI Systems require *activity consideration* to provide precise parametric statistical based profiling.

From our validation, we can conclude that activity is the most contributing factor in our Device model, followed by the Visual Space contribution. If these two parameters are not considered in the model, then it is not possible to use parametric statistical based profiling in 3D TI Systems as the variability in terms of QoS requirements is too significant to provide a precise bound to the Frame Size and the Period that does not underestimate or overestimate the observed values.

As part of our evaluation of the CDG Model, we evaluated the regression model for the Instream and Outstream functions in the CDG. We compare these values with a trace of another participant. The activity performed by the participant was walking. Figure 9.9 shows the regression of the Computation Time for the Instream functionality in a CDG and Figure 9.10 shows the regression of the Computation Time Outstream functionality in a CDG. To validate our regressions we use the Coefficient of Determination R^2, which is a measurement of how well the data points fit a particular function. Our results indicate that the Coefficient of Determination R^2 for the Outstream regression when compared to the trace is 0.697. For the case of the Instream, the R^2 is significantly low at 0.17. We attribute this oddly small value to the high variability in the Computational Demand of the Instream function, especially for the case of small frame sizes. The Instream function is mainly an interrupt driven function as it receives packets from the network. However, the Outstream is significantly more time consuming and has a higher contribution in the overall model than

the Instream, and therefore, these high variations have little effect in the overall model.

Finally, as part of our evaluation, we compare the Device Model with other estimation approaches. We compare our model with the Moving Average adaptation approach used in iDSRT [46] with a window of size 10 for both the frame size and the period. Figure 9.11 shows a trace of 300 frames for the Standing activity. We can observe that moving average tries to minimize the distance between all the points in the sliding window, however, it cannot adapt fast enough to the changes of the weakly period behavior of 3D TI stereo cameras. Therefore, the moving average approach is unsuitable as many of the frame sizes and rates will be underestimated. Our model is more adequate for estimating QoS as the quality parameter can bound the overestimation and provide a probabilistic guarantee on the upper bound in the number of samples underestimated.

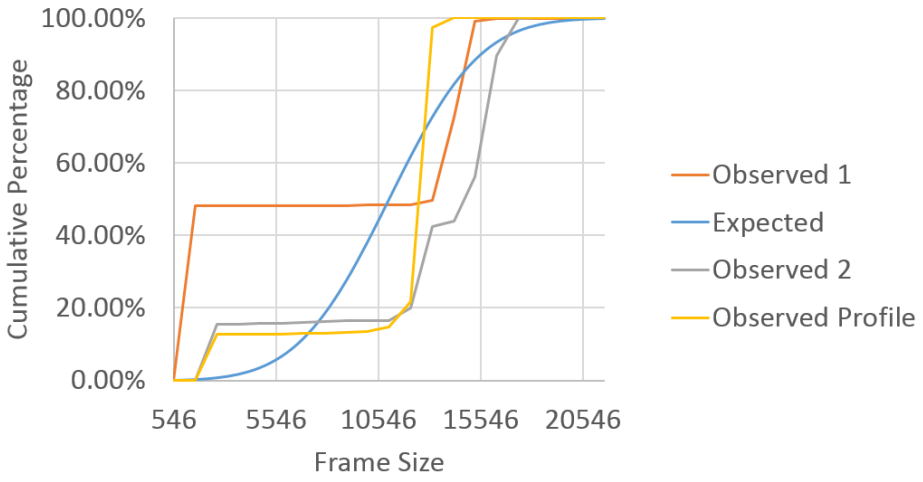
Also, we compared the Device Model with using the median to estimate the frame size and the frame rate. Figure 9.12 shows a trace of 300 frames for the Standing activity. We can observe that the problem with the median is that half of the frames are above the observed values and therefore this approach is a very bad fit for an estimation of the QoS parameters. One advantage of our model is that the quality parameter can estimate the values with different levels of QoS and therefore, it allows to obtain QoS parameters with statistical characteristics similar to the median.

9.4 Conclusion

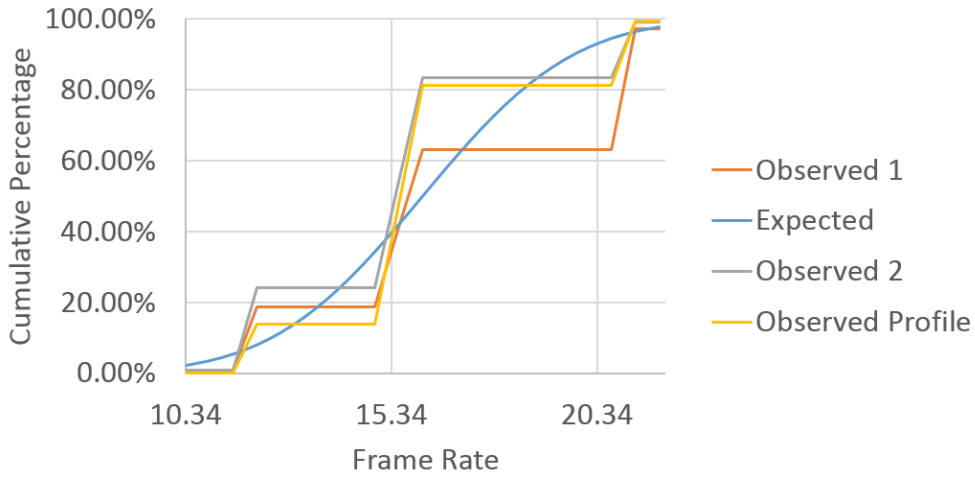
As part of our contribution, we introduce a QoS model for processing and dissemination of streams in 3D TI Systems in which the activity is the main driving factor of the QoS requirements in 3D TI Systems. Our model considers that the activity determines the type of devices and the type of stream processing required in 3D TI systems. Our QoS model considers the Visual Space of a 3D TI Session as part of the variability of the 3D TI activity. It considers the concept of Dominant and Non-Dominant Streams to address the challenges of profiling time and space correlated streams. As part of our model, we also consider activity-driven processing functions of streams, which are an important part of 3D TI Systems.

We use this model to design Hera, an offline profiling tool that allows users to determine the QoS requirements in terms of CPU and bandwidth of a 3D TI session. Hera provides estimation of QoS parameters based on previous statistical data and reduces the amount of over-provisioned resources that must be reserved to meet the QoS requirements of the 3D TI session. We have shown that our model allows to profile the QoS requirements for users in 3D TI based on their activity and their Visual Space Contribution. Our model shows potential as some activities patterns are easy to fit in the model, however some other activities show significant variability in terms of Computation Requirements,

Period and Frame Sizes of the Streams. Overall, Hera reduces the amount of over-provisioning to as little as 5%.

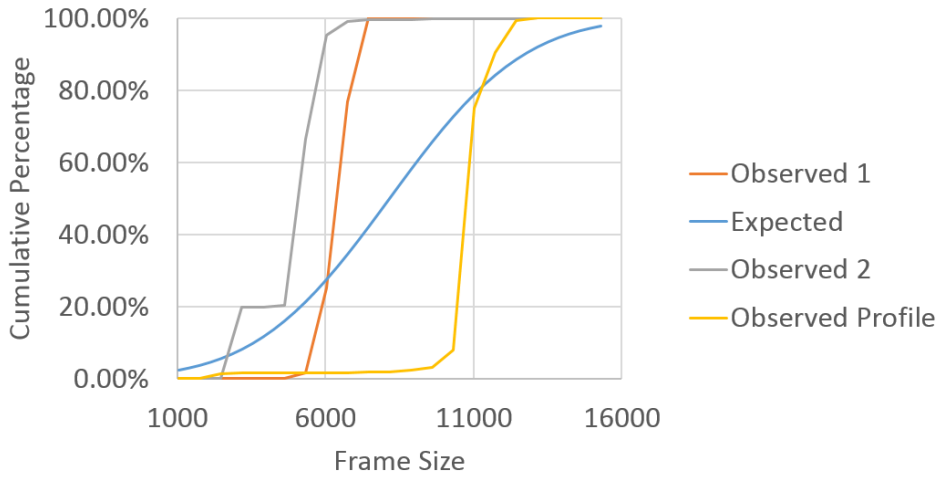


(a) Frame Size Model Evaluation

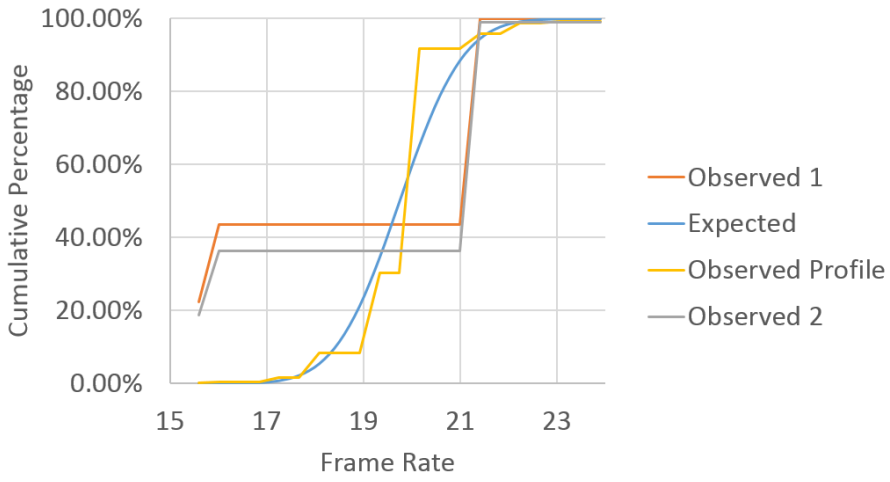


(b) Frame Rate Model Evaluation

Figure 9.5: QoS Device Model Evaluation for Standing Activity with Red Visual Space

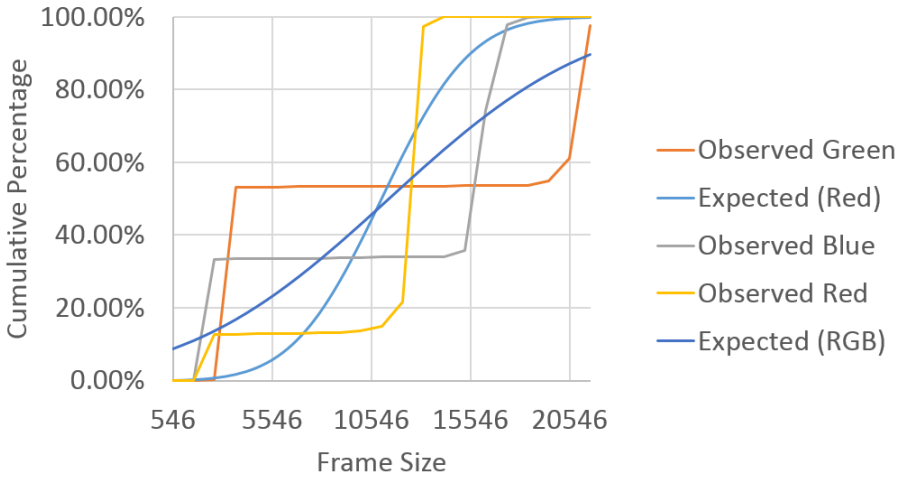


(a) Frame Size Model Evaluation

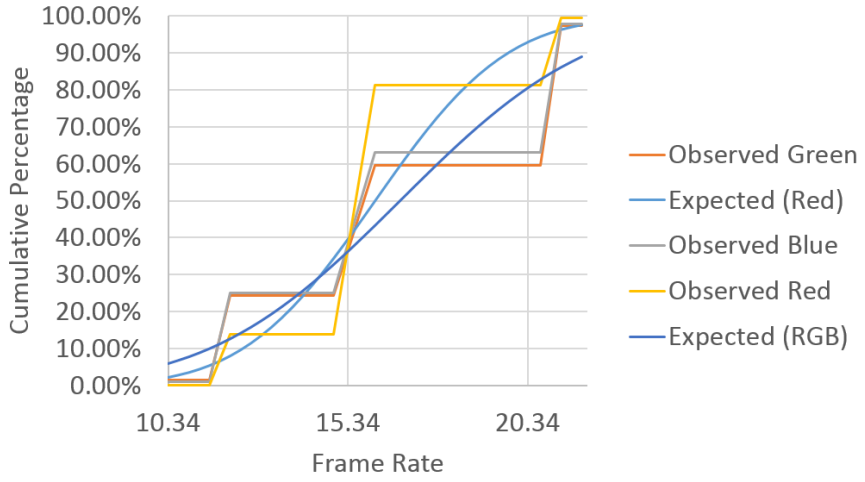


(b) Frame Rate Model Evaluation

Figure 9.6: QoS Device Model Evaluation for Sitting Activity with Red Visual Space

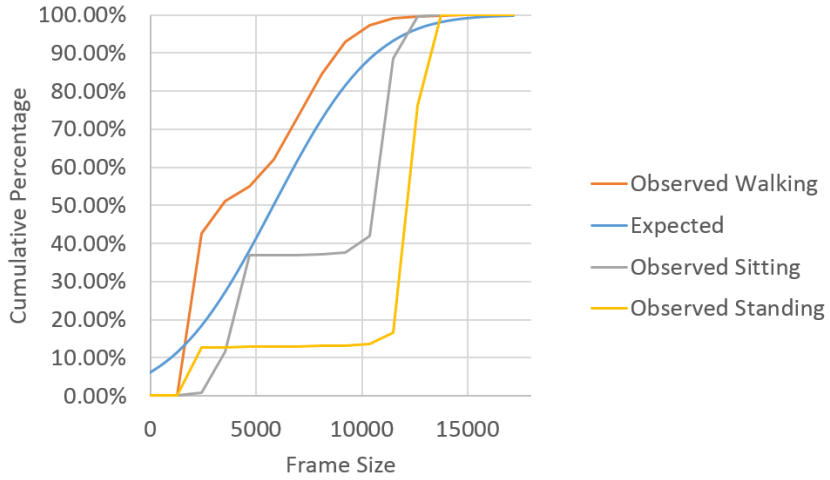


(a) Frame Size Model for Standing Activity

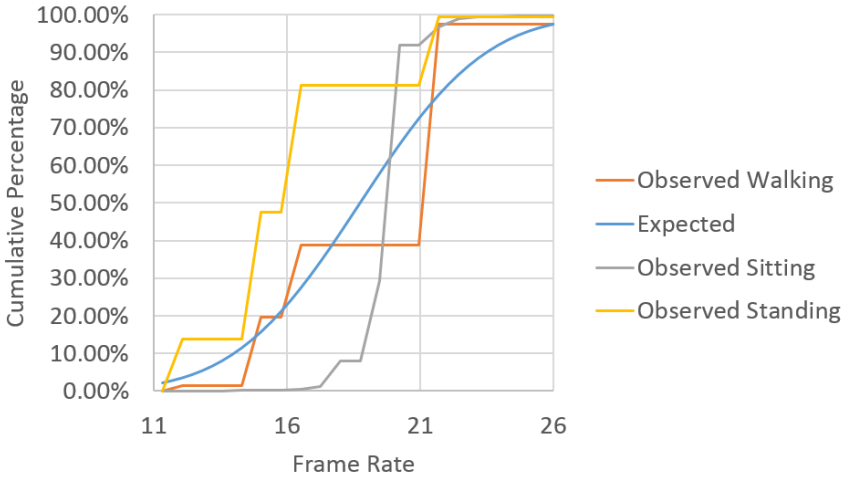


(b) Frame Rate Model for Standing Activity

Figure 9.7: QoS Device Model Evaluation with Multi-Color Visual Space



(a) Frame Size Model without activity consideration



(b) Frame Rate Model without activity consideration

Figure 9.8: QoS Device Model Evaluation without Activity Profiling

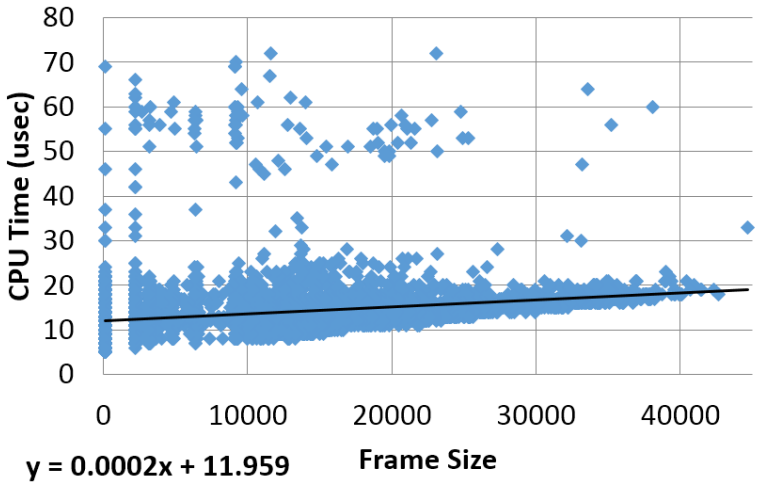


Figure 9.9: Instream Regression

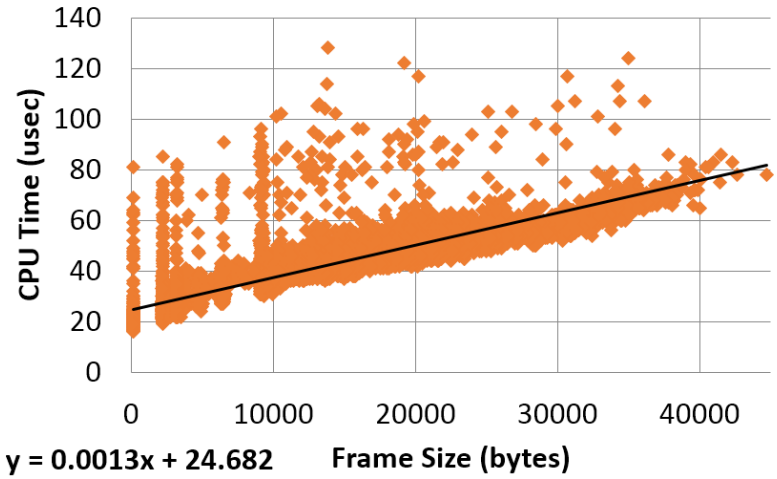
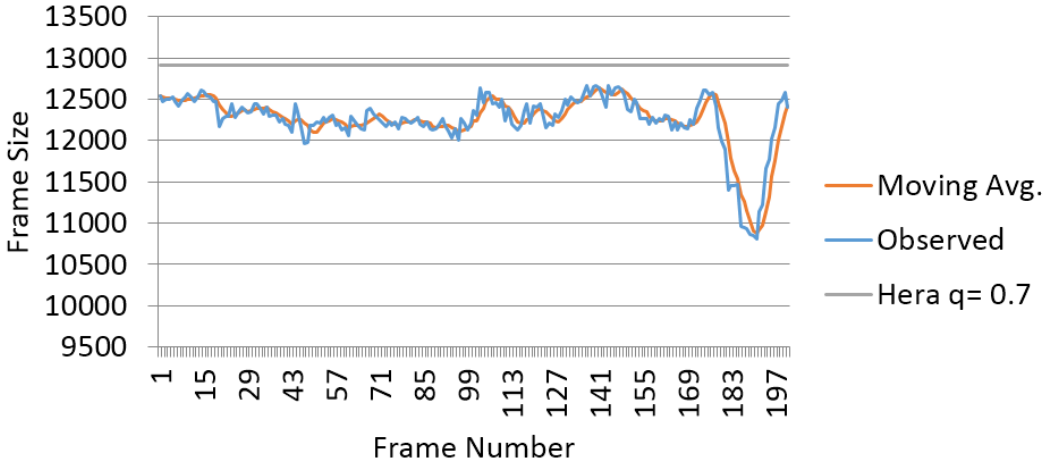
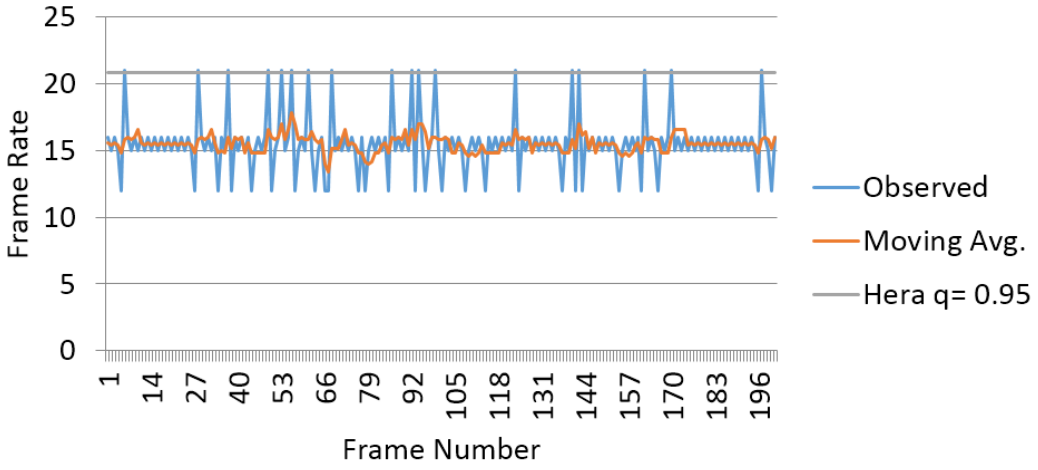


Figure 9.10: Outstream Regression

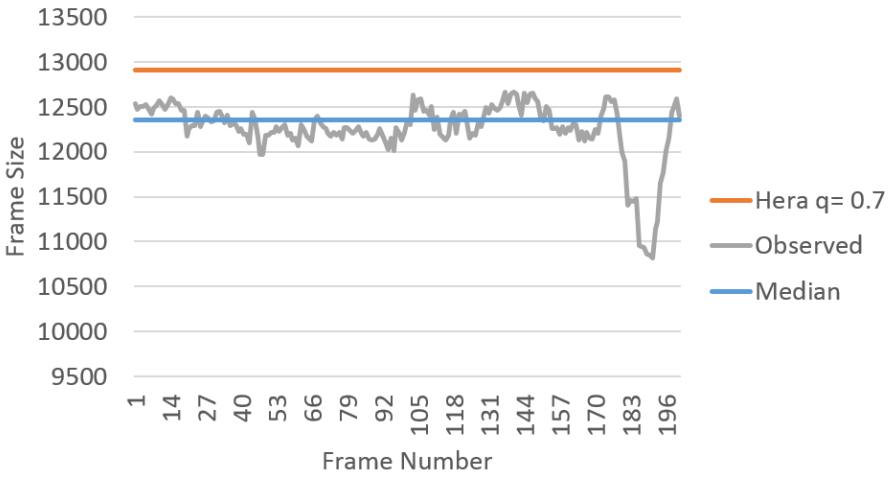


(a) Frame Size comparison

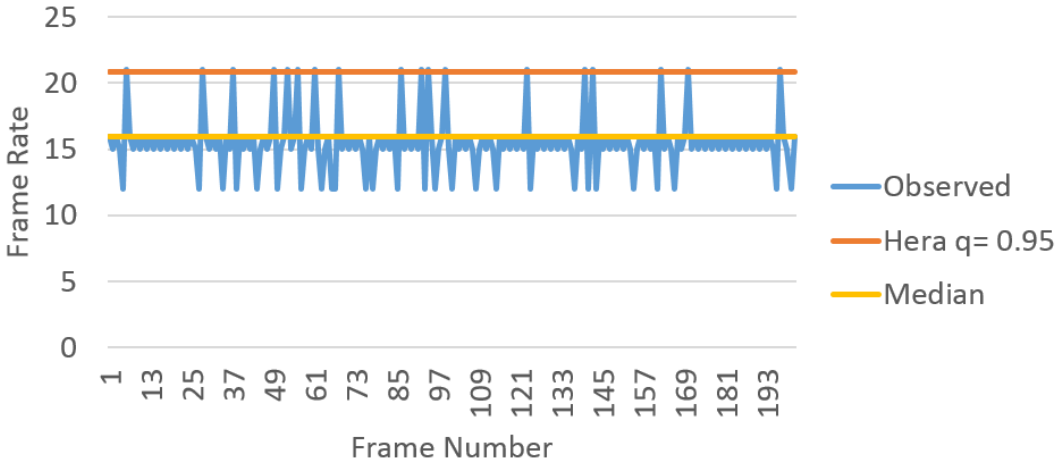


(b) Frame Rate comparison

Figure 9.11: QoS Device Model Comparison with the Moving Average approach for Standing Activity



(a) Frame Size comparison



(b) Frame Rate comparison

Figure 9.12: QoS Device Model Comparison with the Median approach for Standing Activity

Chapter 10

Conclusion and Future Work

In the past few years, 3D Teleimmersive Systems have become ubiquitous and the applications for this systems have exploded in a wide range of directions including telemedicine, distant learning, exergaming and dancing. Research in these areas has seen significant improvements in terms of video quality, Quality of Service(QoS) such as decreased delay and higher video frame rates and higher Quality of Experience (QoE). Approaches that improve session management, monitoring, system management and integration have emerged. In this dissertation we present StreamOS, a Distributed Operating System that provides a holistic substrate of soft real-time resource and device management services and protocols in 3D Teleimmersive Systems.

10.1 Thesis Achievement

Our main contribution is the holistic approach to resource and device management, driven by the activities performed by the participants in 3D Teleimmersive Systems. In this holistic approach, we take into account activities, users and inherent characteristics of the 3D TI Systems impose dependencies on the resource and device management that must be considered across all the components of StreamOS:

- At the device level, the resource management is driven by the activity as different activities determine which devices will be used.
- At the stream level, as different activities, devices and topologies of the sites, create processing pipelines and time dependencies across streams and Bundle of Streams.
- At the task level, as different streams are mapped into groups of dependent and concurrent tasks with QoS parameters in terms of Computation Time and Period.
- At the job level, as frames from correlated streams must be scheduled concurrently to minimize skew across this streams.

We believe that this holistic approach is central to make interactive 3D TI Systems successful. Time and Space coordinated resource and device management

is necessary to provide a coordinated set of policies and mechanisms shared across all the components in the 3D TI System. In addition to our main contribution, our thesis also contributes in several other sub-areas. Below we present some of our contributions.

10.1.1 Correlated Stream Soft Real-Time scheduling

As part of our contribution we introduce a Process Calculus to model the relations of concurrency and dependency between time and space correlated streams in 3D TI Systems. This novel model provides powerful notation that allows modeling the complex constraints of groups of streams in 3D TI Systems. We believe that our model will contribute to the advancement in the area of real-time co-scheduling as future work can leverage this mathematical model to further understand and model stream and task concurrencies and dependencies not only in 3D TI Systems but in other correlated multi-stream systems as well. Our model can be used to simulate systems with concurrencies and dependencies to validate new scheduling algorithms or to find bugs in implementations of other commonly used co-scheduling algorithms.

Our second contribution in this area is Zeus, a soft real-time CPU scheduling architecture for 3D TI Content Delivery Gateways (CDGs) to support Bundle of Streams on multi-core architectures. In Zeus, we use our Process Calculus as a basis to design a novel scheduling algorithm for concurrent and codependent tasks in multi-processor systems based on the partitioned Earliest Deadline First algorithm [26]. Our novel scheduling algorithm uses a concurrency budget based on the laxity of the task (i.e., residual budget) to minimize the amount of the skew between tasks depending on their actual running time. As part of our contribution, we introduce an admission control for group of streams.

We can summarize our contributions in the area of Correlated Soft Real-Time Scheduling as follows:

1. A novel process calculus that simplifies the specification and analysis of dependencies and concurrencies in time and space correlated process and stream systems.

2. Novel algorithms that provide scheduling for concurrent and codependent streams based on multi-core EDF

3. Online algorithms to bound the skew between concurrent tasks based on the actual laxity of the task as computed at run-time.

10.1.2 Activity-based QoS Management for 3D TI Systems

As part of our contribution, we introduce a QoS model for processing and dissemination of streams in 3D TI Systems in which the activity is the main driving

factor of the QoS requirements in 3D TI Systems. Our model considers that the activity determines the type of devices and the type of stream processing required in 3D TI systems. Our QoS model also considers the Visual Space of a 3D TI Session as part of the variability of the 3D TI activity and also considers the concept of Dominant and Non-Dominant Streams to address the challenges of profiling time and space correlated streams. As part of our model, we also consider activity-driven processing functions, which is an important characteristic of 3D TI Systems.

We use this model to design Kratos, a Support Vector Machine to perform activity detection based on 3D TI system metadata. Our approach uses system metadata and therefore it does not rely on specific device formats or interfaces, addressing the problem of detecting an activity in a system with a large number of multimodal interfaces. Also, our approach is fast and therefore is suitable for fast-paced interactive 3D TI sessions. Finally our approach is extensible to other activities making it suitable for 3D TI systems where activities are diverse.

Also, we use this model to design an offline profiling tool based on parametric estimation to obtain probabilistic bounds for the QoS requirements of a 3D TI session based on a particular activity.Hera provides estimation of QoS parameters based on previous statistical data and reduces the amount of over-provisioned resources that must be reserved to meet the QoS requirements of the 3D TI session.

In summary, as part of our contribution in this area we provide:

1. An activity-based QoS model for processing and dissemination of streams in 3D TI Systems in which the activity and the Visual Space (i.e., color and stream dominance) are the main driving factor of the QoS requirements.

2. A novel system based on a Support Vector Machine to perform activity detection based on 3D TI System metadata.

3. An architecture that provides profiling of 3D TI sessions and simplifies the process of determining the QoS requirements of a 3D TI session.

10.1.3 Universal Stream Management and Interface

As part of our contribution, we introduce the Streaming as a Service concept in which real-time data streaming is provided between input and output devices as a transparent layer. In this model, access to disseminating infrastructures is provided through a universal interface in which multimodal devices require no source code modification to interface and instead they provide a specification about their streaming protocols. Our paradigm introduces the concept of groups of time and space correlated streams (i.e., Bundle of Streams [27]) as first class objects.

We also introduce data streaming protocol (S-RTP) based on hierarchical uniform naming for devices, sites and session in 3D TI Systems. Our hierarchical

uniform naming solves the problem of location-context preservation in 3D TI and Telepresence Systems as it allows to uniquely associate a device with a site and a session. We believe that this naming should have an impact in future standards and in currently proposed standards for interoperability between 3D TI Systems (e.g., Clue [67]).

Our third contribution in this area is Prometheus, a streaming framework in which input and output devices stream data through networking tunnels without source modification. Our streaming framework enables location context-based real-time processing of user-defined functions over correlated streams.

In summary, our contributions in this area are:

1. Formalization of the Streaming as Service model.

2. Hierarchical Uniform Naming for 3D TI Systems that allows to preserve the location-context information of devices and streams with respect of which room and session they belong to.

3. A streaming protocol for real-time data delivery (S-RTP) that uses our Hierarchical Uniform Naming to stream groups of correlated streams (i.e., Bundles of Streams).

4. Design of the Prometheus streaming framework that provides streams and bundles as first class objects, unified interface for multimodal end-devices, user-controlled run-time functions over streams and bundles and integrated management for Bundle of Streams.

5. A dual virtualization architecture that enables seamless dynamism of multimodal devices and allows seamless dynamism of activities

10.1.4 Universal Device I/O Management

As part of our contribution, we introduce device and resource naming protocols based on a Hierarchical Uniform Device Identifier for 3D TI Systems that addresses the challenge of preserving the location-context information of devices and streams with respect of which room and session they belong to. Our hierarchical uniform naming solves the problem of location-context preservation in 3D TI and Telepresence Systems as it allows to uniquely associate a device with a site and a session.

Our second contribution is to enable automation of I/O management in TI activities through novel dual virtualization architecture that enables seamless dynamism of devices to address the challenge of heterogeneous non-standard I/O devices, i.e., hides against changes in hardware interfaces in multimodal devices (e.g., if the 3D camera used changes from Bumblebee to Kinect) and allows seamless dynamism of activities to address the challenge of contingency and diversity of activities in 3D TI Systems, i.e., enables universal interface of

activities to distributed I/O, without any concerns of the underlying software changes (e.g., if user activity changes from walking to sitting).

In summary, our contributions in this area are:

1. Device and Resource naming protocols based on a Hierarchical Uniform Device Identifier for 3D TI Systems that addresses the challenge of preserving the location-context information of devices and streams with respect of which room and session they belong to.

2. A dual virtualization architecture that addresses the challenge of enables seamless dynamism of multimodal devices, i.e., hides against changes in hardware interfaces in multimodal devices (e.g., if the 3D camera used changes from Bumblebee to Kinect) and allows seamless dynamism of activities, i.e., enables universal interface of activities to distributed I/O, without any concerns of the underlying software changes (e.g., if user activity changes from walking to sitting).

10.2 Future Work and Lessons Learnt

Resource Management is a very fast paced area. The need for more bandwidth and more resources will always overcome the new network technologies and architecture standards. As more bandwidth becomes available due to advances in telecommunication and more computing power becomes available due to faster and more efficient architectures, ideas of richer and more sophisticated applications will appear.

We have moved from Videoconferencing systems and on-demand audio streaming to rich 3D Teleimmersive systems with stereo cameras that create 3D models of the participants as multiple sensor and haptic devices try to recreate the surrounding environment. StreamOS significantly simplifies resource management in this new generation of 3D Teleimmersion Systems, however, as new and richer applications emerge also resource management should move forward. Here, we discuss some interesting research directions that are open in this domain.

Activity-Based QoS Adaptation: Hera provides a static QoS model for 3D TI based on the activity and the visual contribution of the participants in the 3D TI. However, patterns in activity are very varied and static QoS cannot always provide the best resource management. A system that dynamically adapts the QoS based on changes in pattern in the activity and the visual contribution should improve the overall resource management. For example, this issue arises when a second participant enters the 3D TI space of a session already in progress.

Network Delay Adaptation Support: Zeus provides soft real-time guarantees for time and space correlated streams in Content Delivery Gateways, however Zeus does not account for variability in traffic across multiple gateways.

As 3D TI Systems are deployed across large geographical distances variability in the network conditions become very significant.

References

[1] *Skype.* Microsoft Corp. [Online]. Available: http://www.skype.com

[2] *NetMeeting.* Microsoft Corp.

[3] *Cisco TelePresence Multipoint Switch.* Cisco Systems Inc, 2010. [Online]. Available: http://www.cisco.com/en/US/products/ps7315/index.html

[4] *HP Halo Gateway.* Hewlett-Packard Development Company, L.P, 2007.

[5] O. Schreer, N. Brandenburg, S. Askar, and M. Trucco, "A virtual 3d video-conference system providing semi-immersive telepresence: A real-time solution in hardware and software," in *In Proceedings of eBusiness and eWork*, 2001.

[6] C. C. Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, "The cave: audio visual experience automatic virtual environment," *Commun. ACM*, vol. 35, no. 6, pp. 64–72, February 1992. [Online]. Available: http://dx.doi.org/10.1145/129888.129892

[7] Z. Yang et al., "Teeve: The next generation architecture for tele-immersive environment," in *ISM*. IEEE Computer Society, 2005. [Online]. Available: http://dx.doi.org/10.1109/ISM.2005.113

[8] H. H. Baker, D. Tanguay, I. Sobel, D. Gelb, M. E. Goss, W. B. Culbertson, and T. Malzbender, "The coliseum immersive teleconferencing system," 2002.

[9] W. Wu, M. A. Arefin, Z. Huang, P. Agarwal, S. Shi, R. Rivas, and K. Nahrstedt, ""i'm the jedi!" - a case study of user experience in 3d tele-immersive gaming," in *ISM*, 2010, pp. 220–227.

[10] G. Kurrilo et al., "Immersive 3d environment for remote collaboration and training of physical activities," in *VR*, 2008. [Online]. Available: http://dx.doi.org/10.1109/VR.2008.4480795

[11] M. Tamai, W. Wu, K. Nahrstedt, and K. Yasumoto, "View control interface for 3d tele-immersive environments," in *ICME*, 2008, pp. 1101–1104.

[12] Z. Zhang, D. Chu, J. Qiu, and T. Moscibroda, "Demo: Sword fight with smartphones," in *Proc. of SenSys*, 2011.

[13] W. Wu, A. Arefin, R. Rivas, K. Nahrstedt, R. Sheppard, and Z. Yang, "Quality of experience in distributed interactive multimedia environments: Toward a theoretical framework," in *Proceedings of the 17th ACM International Conference on Multimedia*, ser. MM '09. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1631272.1631338 pp. 481–490.

[14] A. Arefin, Z. Huang, R. Rivas, S. Shi, P. Xia, K. Nahrstedt, W. Wu, G. Kurillo, and R. Bajcsy, "Classification and analysis of 3d teleimmersive activities," *MultiMedia, IEEE*, vol. 20, no. 1, pp. 38–48, 2013.

[15] *Bumblebee2 CCD Stereo Camera Datasheet.* Point Grey Research Inc.

[16] *Direct Show Documentation.* Microsoft. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms783323.aspx

[17] D. Tanguay, D. Gelb, and H. H. Baker, "Nizza: A framework for developing real-time streaming multimedia applications," 2004.

[18] *Distributed Component Object Model (DCOM) Remote Protocol Specification.* Microsoft. [Online]. Available: http://msdn.microsoft.com/library/cc201989.aspx

[19] *Common Object Request Broker Archictecture.* OMG, 1995.

[20] A. S. Tanenbaum and S. J. Mullender, "An overview of the amoeba distributed operating system," *ACM SIGOPS Operating Systems Review*, vol. 15, no. 3, pp. 51–64, 1981. [Online]. Available: http://doc.utwente.nl/55923/

[21] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Lonard, and W. Neuhauser, "Overview of the chorus distributed operating systems," *Computing Systems*, vol. 1, pp. 39–69, 1991.

[22] M. B. Jones, P. J. Leach, R. P. Draves, and I. Barrera, "Modular real-time resource management in the rialto operating system," in *HOTOS '95: Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V).* Washington, DC, USA: IEEE Computer Society, 1995. [Online]. Available: http://portal.acm.org/citation.cfm?id=822074.822393 pp. 12+.

[23] H.-H. Chu and K. Nahrstedt, "Cpu service classes for multimedia applications," in *Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'99)*, vol. 1, 1999. [Online]. Available: http://citeseer.ist.psu.edu/old/698695.html pp. 296–301.

[24] ExpressLogic, "Real-time operating system for embedded development."

[25] J. Nakajima, M. Yazaki, and H. Matsumoto, "Multimedia/realtime extensions for the mach operating system." in *USENIX Summer*, 1991, pp. 183–198.

[26] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," 1973.

[27] P. Agarwal et al., "Bundle of streams: Concept and evaluation in distributed interactive multimedia environments," in *ISM*, 2010.

[28] *Multimedia Class Scheduler Service.* Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/ms684247(v=vs.85).aspx

[29] *GStreamer.* The GStreamer Team. [Online]. Available: http://gstreamer.freedesktop.org/

[30] L. Poettering, *PulseAudio System.* [Online]. Available: http://www.pulseaudio.org

[31] R. Rashid, D. Julin, D. Orr, R. Sanzi, R. Baron, A. Forin, D. Golub, and M. Jones, "Mach: a system software kernel," in *COMPCON Spring '89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers.*, 1989, pp. 176–178.

[32] U. Ramachandran, R. S. Nikhil, J. M. Rehg, Y. Angelov, A. Paul, S. Adhikari, K. M. Mackenzie, N. Harel, and K. Knobe, "Stampede: A cluster programming middleware for interactive stream-oriented applications," 2003.

[33] J. M. Rehg, U. Ramachandran, R. H. Halstead, C. F. Joerg, L. Kontothanassis, R. S. Nikhil, and S. B. Kang, "Space-time memory: A parallel programming abstraction for dynamic vision applications," Tech. Rep., 1997.

[34] R. C. Anand and K. Nahrstedt, "A middleware infrastructure for active spaces," *IEEE Pervasive Computing*, pp. 74–83, 2002.

[35] L. Capra, W. Emmerich, and C. Mascolo, "Carisma: Context-aware reflective middleware system for mobile applications," *IEEE Transactions on Software Engineering*, vol. 29, pp. 929–945, 2003.

[36] D. Engler et al., "Exokernel: an operating system architecture for application-level resource management," in *SOSP*, 1995. [Online]. Available: http://doi.acm.org/10.1145/224056.224076

[37] G. Zhenyu et al., "R2: An application-level kernel for record and replay," in *OSDI*, 2008.

[38] G. Banga et al., "Resource containers: a new facility for resource management in server systems," in *OSDI*, 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=296806.296810

[39] *CoreOS: Linux for Massive Server Deployments.* CoreOS Development Group, 2013.

[40] D. Marinescu and K. Wang, "Gang scheduling and demand paging," in *Proc. of the Int. Conf. on High Performance Computing*, 1995.

[41] J. Ousterhout, "Scheduling techniques for concurrent systems," in *Proceedings of 3rd International Conference on Distributed Computing Systems*, 1982, pp. 22–30.

[42] D. G. Feitelson and L. Rudolph, "Gang scheduling performance benefits for fine-grain synchronization," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 306–318, 1992.

[43] E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fernandez, "Adaptive parallel job scheduling with flexible coscheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, pp. 1066–1077, November 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1092711.1092783

[44] M. Barabanov and V. Yodaiken, "Introducing real-time linux," *Linux Journal*, no. 34, February 1997.

[45] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource kernels: A resource-centric approach to real-time and multimedia systems," in *In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, 1998, pp. 150–164.

[46] H. Nguyen, R. Rivas, and K. Nahrstedt, "iDSRT: Integrated dynamic soft real-time architecture for critical infrastructure data delivery over WLAN," *Mobile Networks and Applications*, June 2010. [Online]. Available: http://dx.doi.org/10.1007/s11036-010-0250-x

[47] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *Proc. of 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003. [Online]. Available: http://citeseer.ist.psu.edu/yuan03energyefficient.html

[48] T. Cucinotta, F. Checconi, L. Abeni, and L. Palopoli, "Self-tuning schedulers for legacy real-time applications," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755921 pp. 55–68.

[49] J. Nieh and M. S. Lam, "The design of smart: A scheduler for multimedia applications," 1996.

[50] S. Kato and Y. Ishikawa, "Gang edf scheduling of parallel task systems," in *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, ser. RTSS '09. Washington, DC, USA: IEEE Computer Society, 2009. [Online]. Available: http://dx.doi.org/10.1109/RTSS.2009.42 pp. 459–468.

[51] W. Yuan and K. Nahrstedt, "Process group management in cross-layer adaptation," in *In Proc. of SPIE/ACM Multimedia Computing and Networking Conference*, 2004, pp. 55–68.

[52] K. Lakshmanan, D. de Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," *Real-Time Systems Symposium, IEEE International*, vol. 0, pp. 469–478, 2009.

[53] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes," *Information and Computation*, vol. 100, no. 1, pp. 1 – 40, 1992.

[54] "Iso standard 8807, lotos, a formal description technique based on temporal ordering of observational behavior," 1989.

[55] S. Roy et al., "A system architecture for managing mobile streaming mediaservices," *Distributed Computing Systems*, vol. 0, p. 408, 2003.

[56] P. Schramm et al., "A service gateway for networked sensor systems," *Pervasive Computing*, 2004.

[57] D. Valtchev et al., "Service gateway architecture for a smart home," *IEEE Commun.*, vol. 40, no. 4, pp. 126 –132, 2002.

[58] M. Weihs, "Design issues for multimedia streaming gateways," *Mobile Communications and Learning Technologies*, vol. 0, p. 101, 2006.

[59] W. Wu et al., "Implementing a distributed tele-immersive system," in *ISM*, 2008.

[60] D. Marples et al., "The open services gateway initiative: an introductory overview," *IEEE Commun.*, vol. 39, no. 12, pp. 110 –114, 2001.

[61] H. Cervantes et al., "Beanome: A component model for the osgi framework," in *Software infrastructures for component-based applications on consumer devices*, 2002.

[62] Y. E. Sung et al., "Modeling and understanding end-to-end class of service policies in operational networks," in *SIGCOMM*, 2009. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592595

[63] X. Zhang, J. Liu, B. Li, and T. Yum, "Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, 2005, pp. 2102–2111 vol. 3.

[64] F. Wang, Y. Xiong, and J. Liu, "mtreebone: A collaborative tree-mesh overlay network for multicast video streaming," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 3, pp. 379–392, 2010.

[65] M. P. E. Group, "Dynamic adaptive streaming over http (dash)," *ISO/IEC 23009-1*, 2009.

[66] H. Alvestrand, "Real time protocols for brower-based applications," 2013.

[67] *ControLling mUltiple streams for tElepresence (CLUE).* The CLUE Workgroup. [Online]. Available: https://www2.ietf.org/wg/clue

[68] *InfoSphere Streams.* IBM, 2009.

[69] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "A case for a coordinated internet video control plane," in *ACM SIGCOMM*, 2012.

[70] *Akamai HD Network.* Akamai. [Online]. Available: http://www.akamai.com/html/misc/hdnetwork.html

[71] *Microsoft Smooth Streaming.* Microsoft. [Online]. Available: http://www.microsoft.com/silverlight/smoothstreaming/

[72] *Quicktime for Developers.* Apple. [Online]. Available: https://developer.apple.com/quicktime/

[73] *InfiniBand Architecture Specification 1.2.1.* InfiniBand Trade Association, 2007.

[74] J. Satran, K. Meth, and C. Sapuntzakis, "Rfc 3270: Internet small computer systems interface," 2004.

[75] *Universal Serial Bus Specification 3.0.* USB Implementers Forum, Inc, 2011.

[76] "Ieee standard for a high-performance serial bus," *IEEE Std 1394-2008*, pp. 1 –906, 21 2008.

[77] *SCSI Architecture Model (ANSI X3.270-1996).* American National Standards Institute, 1996.

[78] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003. [Online]. Available: http://dx.doi.org/10.1145/945445.945462 pp. 164–177.

[79] C. A. Waldspurger, "Memory resource management in vmware esx server," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, 2002. [Online]. Available: http://dx.doi.org/10.1145/844128.844146

[80] *Server Message Block (SMB) Version 2 Protocol Specification*. Microsoft Corp, 2012.

[81] S. Shepler, B. Callaghan, and D. Robinson, "Rfc 3530: Network file system (nfs) version 4 protocol," 2003.

[82] A. Rubini, "The virtual file system in linux," *Linux Journal*, 1997.

[83] D. R. Cheriton, "Uio: a uniform i/o system interface for distributed systems," *ACM Trans. Comput. Syst.*, vol. 5, no. 1, pp. 12–46, Jan. 1987. [Online]. Available: http://doi.acm.org/10.1145/7351.7353

[84] K.-D. Moon, Y.-H. Lee, C.-E. Lee, and Y.-S. Son, "Design of a universal middleware bridge for device interoperability in heterogeneous home network middleware," *Consumer Electronics, IEEE Transactions on*, vol. 51, no. 1, pp. 314 – 318, feb. 2005.

[85] V. Miori, L. Tarrini, M. Manca, and Tolomei, "Domonet: a framework and a prototype for interoperability of domotic middlewares based on xml and web services," in *Consumer Electronics, 2006. ICCE '06. 2006 Digest of Technical Papers. International Conference on*, jan. 2006, pp. 117 – 118.

[86] I. Foster, D. Kohr, Jr., R. Krishnaiyer, and J. Mogill, "Remote i/o: Fast access to distant storage," in *In Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*. ACM Press, 1997, pp. 14–25.

[87] N. K. Sharma, D. E. Irwin, P. J. Shenoy, and M. Zink, "Multisense: fine-grained multiplexing for steerable camera sensor networks," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943556 pp. 23–34.

[88] W. Niu, J. Long, D. Han, and Y.-F. Wang, "Human activity detection and recognition for video surveillance," in *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, vol. 1, 2004, pp. 719–722 Vol.1.

[89] J. Sung, C. Ponce, B. Selman, and A. Saxena, "Unstructured human activity detection from rgbd images." in *ICRA*. IEEE, 2012, pp. 842–849.

[90] D. Peebles, H. Lu, N. D. Lane, T. Choudhury, and A. T. Campbell, "Community-guided learning: Exploiting mobile sensor users to model human behavior." in *AAAI*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[91] C.-Y. Chen and K. Grauman, "Efficient activity detection with max-subgraph search," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 1274–1281.

[92] A. Arefin et al., "Classification and analysis of 3d tele-immersive activities," University of Illinois, Urbana-Champaign, Tech. Rep., 2012.

[93] Z. Huang, W. Wu, K. Nahrstedt, A. Arefin, and R. Rivas, "Tsync: a new synchronization framework for multi-site 3d tele-immersion," in *NOSSDAV*. New York, NY, USA: ACM, 2010. [Online]. Available: http://doi.acm.org/10.1145/1806565.1806577 pp. 39–44.

[94] H. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," in *Distributed Computing Systems, 1993., Proceedings the 13th International Conference on*, May 1993, pp. 428 –437.

[95] R. Dahlhaus and M. Eichler, "Causality and graphical models in time series analysis," 2001. [Online]. Available: http://galton.uchicago.edu/˜eichler/hsss.pdf

[96] D. S. Johnson, "Fast algorithms for bin packing," *J. Comput. Syst. Sci.*, vol. 8, pp. 272–314, June 1974. [Online]. Available: http://dx.doi.org/10.1016/S0022-0000(74)80026-7

[97] J. M. López, M. García, J. L. Díaz, and D. F. García, "Utilization bounds for multiprocessor rate-monotonic scheduling," *Real-Time Syst.*, vol. 24, no. 1, pp. 5–28, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1021749005009

[98] T. P. Baker, "A comparison of global and partitioned edf schedulability tests for multiprocessors," Tech. Rep., 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.5515

[99] A. Waterland, "Stress workload generator for posix systems." [Online]. Available: http://people.seas.harvard.edu/ apw/stress

[100] B. Lampson et al., "Reflections on an operating system design," *Commun. ACM*, vol. 19, pp. 251–265, 1976. [Online]. Available: http://doi.acm.org/10.1145/360051.360074

[101] *Google Protocol Buffers API.* Google Inc. [Online]. Available: https://developers.google.com/protocol-buffers

[102] B. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. of COLT*, 1992.

[103] N. Rizzolo and D. Roth, "Learning based java for rapid development of NLP systems," http://bit.ly/13J1kuq, 2010.

[104] A. Keys, F. Fidanza, M. Karvonen, and N. Kimura, "Indices of relative weight and obesity," 1972.

[105] M. A. Arefin, K. Nahrstedt, R. Rivas, J. Han, and Z. Huang, "Diamond: Correlation-based anomaly monitoring daemon for dime." in *ISM*. IEEE Computer Society, 2010, pp. 137–144.

[106] A. Jain, A. Arefin, R. Rivas, C.-n. Chen, and K. Nahrstedt, "3d teleimmersive activity classification based on application-system metadata," in *Proceedings of the 21st ACM international conference on Multimedia*, ser. MM '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2502081.2502194 pp. 745–748.

[107] Z. Yang, B. Yu, K. Nahrstedt, and R. Bajscy, "A multi-stream adaptation framework for bandwidth management in 3d tele-immersion," in *Proceedings of the 2006 International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1378191.1378209 pp. 14:1–14:6.