Stony Brook University

Building Distributed Data Models in a Performance-Optimized, Goal-Oriented Optimization Framework for Cyber-Physical Systems

A Dissertation Presented
by
Varun Subramanian
to
The Department of Electrical and Computer Engineering
in Fulfillment of the Requirements
of the Degree of
Doctor of Philosophy
in
Computer Engineering

Stony Brook University
December 2012

STONY BROOK UNIVERSITY
THE GRADUATE SCHOOL

Varun Subramanian

We, the dissertation committee for the above candidate for the
Doctor of Philosophy degree,
hereby recommend the acceptance of this dissertation.

Alex Doboli, Advisor of Dissertation
Associate Professor, Department of Electrical and Computer Engineering

Xin Wang, Chairperson of Defense
Associate Professor, Department of Electrical and Computer Engineering

Emre Salman, Assistant Professor
Department of Electrical and Computer Engineering

Jennifer Wong, Assistant Professor
Department of Computer Science

This dissertation is accepted by the Graduate School

Charles Taber
Interim Dean of the Graduate School

Abstract of the dissertation

Building Distributed Data Models in a Performance-Optimized, Goal-Oriented Optimization Framework for Cyber-Physical Systems

by
Varun Subramanian
Doctor of Philosophy
in
Computer Engineering
Stony Brook University
2012

Cyber-physical systems (CPS) are large, distributed embedded systems that integrate sensing processing, networking and actuation. Developing CPS applications is currently challenging due to the sheer complexity of the related functionality as well as the broad set of constraints and unknowns that must be tackled during operation. Building accurate data representations that model the behavior of the physical environment by establishing important data correlations and capturing physical laws of the monitored entities is critical for dependable decision making under performance and resource constraints.

The goal of this thesis is to produce reliable data models starting from raw sensor data under tight resource constraints of the execution platform, while satisfying the timing constraints of the application. This objective was achieved through adaptation policy designs that optimally compute the utilization rates of the available network resources to satisfy the performance requirements of the application while tracking physical entities that can be quasi-static or dynamic in nature. The performance requirements are specified using a declarative, high-level specification notation that correspond to timing, precision and resource constraints of the application. Data model parameters are generated by solving differential equations using data sampled over time and modeling errors occur due to missed data correlations and distributed data lumping of the model parameters.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDMENTS

I would like to begin this acknowledgement by thanking my advisor Dr. Alex Doboli who has been my mentor throughout my PhD process. He always found time to solve my queries in his busy schedule. I will always remember the interesting conversations I had with him, the topics ranging from research and academia to entertainment and sports. I really learnt a lot from him and I tried to be an ideal student as he is an ideal adviser.

I also thank my colleagues Cristian Ferent and Anurag Umbarkar and also my ex-colleagues Pengbo Sun and Meng Wang in the Embedded Systems Laboratory for the help they have provided when needed. I would specially like to thank Cristian and Anurag who have been good friends to me and also helped me solve complex research issues. We have also been helpful to each other to ease the stress and tension we go through in the lab by periodically making jokes and having good laughs. It is always good to have good friends like them around you while at work.

I would like to end this acknowledgement by thanking my parents and my beloved sister in India and also my uncle Mr. C.V Kumar who lives here in Long Island. Even though my parents and my sister were not physically present here to help me, they have been very supportive in whatever decisions I made. My uncle was my mentor and guardian throughout my stay here in the United States. He helped me a lot in making rational decisions ever since I decided to pursue my higher studies in the United States.

Chapter 1

Introduction

1.1 Introduction to Cyber Physical Systems

Cyber-physical systems (CPS) are distributed embedded systems that utilize a large variety of sensors, actuators and processing power to integrate the cyber world with the physical world [56]. CPS are rapidly emerging as a main computing paradigm for applications in environmental monitoring, energy conservation, healthcare, infrastructure management, homeland security, intelligent traffic management, manufacturing, and retail [17, 21, 56]. This is due not only to sensing and electronic devices becoming extremely cheap and small in size, thus deployable in large numbers, but also to their potential of offering superior data acquisition and decision making capabilities, if the related aspects are tackled collectively instead of separately. Moreover, CPS are expected to provide better robustness, including responses to unexpected conditions as well as critical situations. Providing efficient and reliable data acquisition for decision making in large, distributed embedded systems currently represents a main challenge [56].

The acquired data is used in decision making to model the behavior of physical entities of interest, like atmosphere, water, soil, radio and sound signals, oil fields, traffic flow, and so on [36, 50, 12, 44]. For example, refer to Figure 1.1 that describes an application scenario in an urban environment where sensor nodes equipped with a variety of sensors are spread across an area and the application goal is to route vehicles along paths that would reduce the overall amount of toxic gas pollution. Real-time data is acquired from three layers of sensors distributed over the region: a gas sensor layer, a temperature sensor layer, and an acoustic sensor layer. The acquired data at

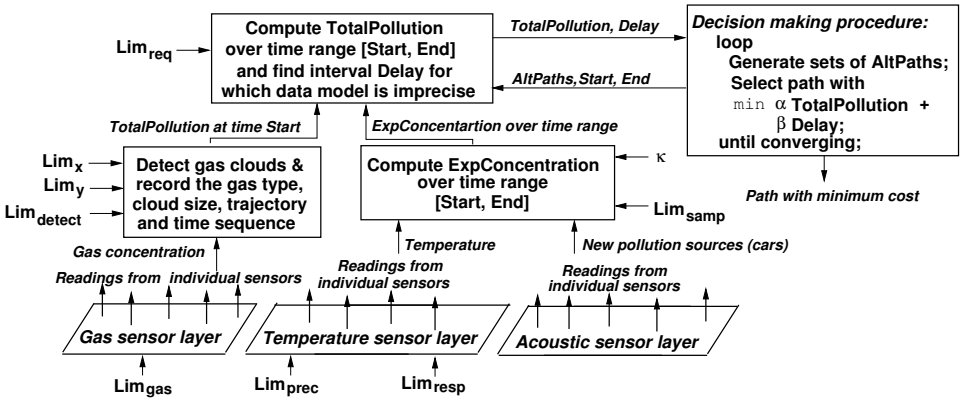

Figure 1.1: Main characteristics of the illustrating example

each node includes position, dynamics (e.g., tracking), and concentration of the toxic gas clouds from gas sensors.

Due to the distributed nature of the physical entities, individual nodes in the application have insufficient data. Hence, these nodes need to collaborate and transform this raw data into a meaningful representation. In the application example, readings from neighboring sensors are aggregated together so that contiguous gas clouds are identified and their size and trajectory can be found in real-time while tracking the cloud dynamics. Aggregated data from sensors can also be transformed into data models. These data models generate model parameters that would describe the manner in which data is distributed in the region, establish data correlations and relations that pertain to physical laws of the monitored entities. In order to obtain the above properties, physical models are built by discretizing the space and introducing state variables for every node in the network. Each of these state variables were expressed in the form of differential equations. For example, the form of differential equations used in this work is given by equation 1.1. The equation is valid for a node i with k neighbors. The only known terms are the Y values that represent the sensor measurements. $Y_i(i)$ corresponds to sensor measurement of node i and $Y_k(t)$ are the sensor measurements of the neighbors of i. The $G_{r_{i,k}}$ terms are the gradient coefficients that model data flow between variables, and the S_{c_i} term is the storage coefficient that indicates the rate of energy stored in node i. The rate of adding or removing energy from the external environment is modeled using $\dot{E}_{in,i}$ and δ is the time difference between consecutive samples. The model parameters that include the $G_{r_{i,k}}$ terms and the S_{c_i} term are computed by solving these differential equations for each node

using sampled data and streamed along the data paths to a common collection point called the target point, which can use this information to make decisions specific to the application goals. Hence, the form of differential equations that are used in this work produce model parameters that model data flow (gradients) between variables and the rate at which energy is stored or dissipated at each node.

$$S_{c_i} \frac{[Y_i(t) - Y_i(t-\delta)]}{\delta} = \sum_k [\frac{Y_k(t) - Y_i(t)}{G_{r_{i,k}}}] + [\dot{E}_{in,i}] \tag{1.1}$$

Considering the example application in Figure 1.1, the $G_{r_{i,k}}$ terms would describe the dynamics of the toxic gas cloud over a region and S_{c_i} terms would describe the level of toxicity (density) of a gas cloud at the corresponding location.

Apart from generating meaningful data models from raw sampled data, CPS applications also need to detect emergence of a physical phenomenon in a distributed area. The creation and behavior of these emergent entities are defined from enabling conditions, that correspond to specifying rules for defining an emergent entity. These rules could define the minimum level of toxicity of an emergent gas cloud and the minimum area covered by the cloud.

The precision of the data models depends on the performance constraints of the embedded infrastructure, including sampling frequency, precision of data acquisition, memory size, communication bandwidth, and power consumption. The performance of the infrastructure conditions the amount of sampled data and the delay with which data becomes available to the decision making routines. The correlations, requirements, and constraints of the application must be precisely specified, so that it can be efficiently tackled under the tight resource constraints of the architecture.

The insight obtained from the process of generating data models are useful while employing decision making procedures. Decision making may include adjusting the network parameters and/or performing control procedures depending on the application needs and resource availability. The application example shown in Figure 1.1 would employ the decision making procedure of finding travel paths for vehicles within the region such that the overall pollution is minimized, e.g., CO, CO_2, SO_2, and NO_x pollution. The characteristics of the polluting clouds constantly change in time and space due to weather conditions (i.e. wind, temperature and humidity) and new pollution sources, such as ongoing traffic. Finding the shortest-length paths does not necessarily minimize the experienced pollution, even though it reduces exposure

time. Instead, decisions about the optimal paths must be continuously made during travel depending on the current and expected levels of the toxic gas concentration. The procedure should generate multiple paths (variable $AltPaths$ in Figure 1.1) and analyze each path according to a cost function that captures two aspects, the total pollution level along the path (variable $TotalPollution$) and the precision of the pollution estimation (variable $Delay$). For example, data received at the target point after a certain time limit (variable Lim_{req}) are less precise in evaluating the total pollution due to the dynamics of gas clouds. Variables $Start$ and $Time$ define the time window over which the expected pollution levels are predicted. Depending on existing conditions and data predictions, the path that is expected to produce the least amount of pollution is selected.

1.2 Design challenges in CPS

The previous section provides an introduction to CPS and details its design challenges using the application example in Figure 1.1.

This section provides a brief description of the characteristics of Cyber Physical Systems and its design challenges that are addressed in this thesis work.

- *Physically-distributed data models*: The data used in decision making is physically distributed in space and time, such as the pollution levels along travel routes and gas clouds placed over 2D zones. Data is utilized to express cumulative descriptions (e.g., size and position of toxic clouds, total amount of pollution), current and predictive trends (i.e. dispersion of clouds in time and space), and integration of data from related sensors, such as gas sensors, temperature sensors and sensors to build data model parameters that describe the phenomenon.

- *Emergent objects*: The forming of a new gas cloud represents an emergent object as the position, time, and gas type of the cloud cannot be anticipated a-priori. In contrast to dynamic variables of procedural languages, which are explicitly created at specific points in a program, emergent objects are produced if their enabling conditions are met, such as a gas cloud is formed if the gas concentration level at all points of a contiguous area exceeds a predefined threshold.

- *Data predictions*: The performed decisions, e.g., the selected travel paths, depend not only on the current state, such as the current pollution level, but also on the future characteristics of the system, i.e. pollution levels at future time instances. The prediction procedures for future characteristics of the monitored entities declare both invariant features, like the physical properties of the entities (e.g., the physical laws of gases), and changing aspects, like weather and traffic conditions, which modify the expressions used in prediction.

- *Application goals and constraints*: The quality of the acquired data is critical for optimal decision making, including sampling sufficient data over space and time, and minimizing the loss and delay of transferring distributed data to the decision making routine given the bandwidth, memory and energy constraints of the architecture. Various timing constraints characterize data acquisition and event handling.

1.3 Objective

The goal of this thesis is to produce reliable data models starting from raw sensor data under tight resource constraints of the execution platform while satisfying the timing constraints of the application.

The above mentioned objective is achieved by following the steps enumerated below:

1. Utilizing the execution support to transfer executable code generated from goal-oriented descriptions to the individual nodes, run network-level applications and detect emergent phenomena based on the rules that are described by the application.

2. Building dynamic optimization policies for network parameters to efficiently track quasi-static physical entities while satisfying performance requirements of the application.

3. Building dynamic optimization policies for network parameters to efficiently track dynamic physical entities while satisfying performance requirements of the application.

4. Building data models that would build meaningful representations of the monitored phenomena and characterizing modeling errors that would occur due to model parameter lumping, and missed data correlations that occur depending on the utilization rates of the data paths.

The first step generates executable code for individual nodes based on goal-oriented descriptions of the application and uses the execution platform to transmit this code to the nodes in the network, before running network-level applications on this execution platform. The execution support also detects the emergence of an entity based on pre-defined thresholds. These thresholds may correspond to the minimum density at each point and the minimum area that has to be covered by the entity. The second and third steps ensure that sufficient data is available at the target point (decision making node in the network) for building highly precise data models by satisfying the timing constraints of the application using resource parameter switching strategies based on the application goals and the nature of the physical entities. The fourth step builds data model parameters using differential equations in the form of equation 1.1 using data sampled from the environment. These parameters model gradients of data flow within the network and the density of sampled data at different points in the network. The effects of lumping these model parameters while transferring them to the target point are studied, and the missed correlations between physical points that are induced by data paths are characterized.

Based on the above description that states the primary objective this thesis and the steps taken to achieve this objective, we can derive the importance of addressing these challenges compared to the existing techniques proposed by the community of Wireless Sensor Networks (WSNs). The main goal of methods proposed for WSNs is to improve the network performance, e.g., bandwidth, throughput, and energy consumption [95, 96, 97, 99]. In contrast, the main goal of the presented work for developing CPS applications is providing accurate data models while meeting all constraints of the application and embedded architecture. Operating at a higher level of abstraction compared to the WSN community, the conducted research work optimizes network parameters while considering performance requirements, like overall timing, precision and resource constraints. Precision constraints are satisfied when sufficient amount of accurate data is sampled by the sensors and also made available at the decision making nodes for correct data representation. Satisfying timing constraints directly relates to reducing latency. Latency corresponds to the delay with which

data is available at the target point. Hence, reducing latency is critical to produce accurate data models that describe the monitored phenomena. Very high latency will result in imprecise descriptions of the the toxic gas clouds in the application example considering the dynamic nature of the toxic gas clouds, making the available data obsolete. A more critical application that requires hard timing constraints is the airbag controller in automobile applications. The sensors need to collaborate quickly and the decision to deploy the airbag has to be made within the precise timing constraints to avoid human injuries. The resource constraints correspond to limited availability of network resources like buffer memory, communication bandwidth, and energy consumption. The challenge is to optimally utilize these limited resources to satisfy all application needs.

The report describes different optimization and algorithmic procedures that tackle these challenges in a systematic manner. Before going into the details of these procedures, Chapter 2 discusses the procedure to specify application goals, procedures and constraints of an application, and also the techniques to transform these specifications into executable code, while Chapter 3 provides the execution platform of a grid network of reconfigurable embedded nodes to run these applications based on the goal-oriented specifications. Chapters 4 and 5 describe the optimization algorithms that use the properties of the monitored entities to select the right network resources based on their availability to satisfy the performance requirements of the applications. In other words, these optimization techniques ensure that sufficient amount of data is available at the target point within the specified timing delay, which is important for building precise data models that can be used for decision making. Chapter 6 describes the procedure of building data model parameters by solving differential equations and streaming these model parameters to the target point. The procedure of distributed data lumping is used where model parameters at different points in the network are lumped for cumulative data representations to reduce communication overhead. The modeling errors that occur during the process of distributed data lumping and the missed data correlations due to the nature of the data paths are characterized in Chapter 6. Chapter 7 concludes the report by summarizing the achieved objectives and also provides future work.

Chapter 2

A High-Level Specification Notation to describe CPS Applications[1]

2.1 Introduction

Specifications at high-levels of abstraction require less development effort as compared to descriptions closer to the physical level [59], and create additional opportunities for optimizing the implementation performance for CPS applications. High-level descriptions mainly focus on expressing the goals, sensing inputs, actuation outputs, events, and constraints of an application, and less on implementation details, such as processing algorithms, routing methods, and data sharing across the distributed platform. This leads to shorter descriptions that are simpler to maintain [59]. Moreover, the burden of tackling complex functionalities and variations in the operating conditions is shifted from the programmer to the compiling and execution support. In traditional specification languages for CPS [25, 53, 15, 30, 29, 27, 47], the programmer has to specify explicitly the processing algorithms, interactions between tasks, data routing, and cluster formation rules. This reduces the opportunities for co-optimizing the network-level data acquisition, transfer, and processing, including dynamic conditions that are hard to predict off-line. In spite of these advantages, there are few high-level programming frameworks for CPS applications.

[1]The work on high-level specification notation was published in [1], and the work on distributed KDS was published in [4]

This chapter presents a novel goal-oriented programming framework for developing CPS applications executed on physically-distributed networks of reconfigurable embedded nodes in section 2.3. The considered applications involve physically distributed data and events, objects emerging dynamically in time and space, and data requiring predictions about their evolution. Such characteristics are typical for many applications in environmental monitoring, infrastructure management, and intelligent traffic management. The proposed framework is based on a declarative, Z-based programming notation. Descriptions express the optimization goals, sensed inputs, actuation outputs, events, and performance constraints, while leaving to the optimizer and execution environment the task of optimally implementing the functionality. Goals and constraints are expressed based on distributed variables that specify data acquired over distributed physical regions. The operators performed on these distributed variables include set-related operators, predicate logic operators, cumulative operators, like integrals, and differentials. Section 2.4 uses the application described in figure 1.1 to provide three examples to specify application goals, the rules to monitor a phenomenon, and the constraints and precision requirements for data sensing. In section 2.5, techniques to compile these goal-oriented descriptions into executable code are discussed, which can be made available at the individual nodes. The executable code is utilized by the execution support to perform network-related operations. The execution support of the networked infrastructure is detailed in Chapter 3.

The distributed variables that are used to make the above goal-oriented descriptions correspond to entities that are distributed in space. These distributed variables are dynamic in nature. This dynamic nature causes different sets of sensor nodes to produce events over time and is a consequence of a number of external factors that govern object movements. For example, consider a scenario where the application goal is to track toxic clouds over an urban area similar to the application example described in Figure 1.1. The nodes are distributed in the form of a grid and are equipped with gas sensors that also have gas classification capabilities. Figure 2.6 shows three local regions that are dynamically formed based on cloud composition. Each region contains gases of different types, including composition and nature. The shaded region is the common region, and is heterogeneous due to the presence of both clouds. Since the cloud is dynamic, it can move in space and also change its properties over time. For instance, the dotted lines in Figure 2.6 indicate that region II has

expanded after a certain time. Consequently, region III will also expand.

This dynamic nature of the distributed variables is represented using a fully decentralized method based on Kinetic Data Structures (KDS) [86]. Unlike the variables or data structures defined in programming languages which are static in nature (i.e. data structures can hold only fixed types like float, integer, etc. and they have a fixed memory assigned to them), Kinetic Data Structures have properties that can vary over time making the concept very suitable to describe variables that are dynamic in nature, i.e. they change their properties over time. The focus of this KDS-based model is representation of distributed variables having the properties of kinetic data on a grid network of wireless embedded sensing nodes once emergent entities is detected. Detecting emergent entities is a capability of the execution platform of the grid sensor network that is described in detail in Chapter 3.

In this KDS-based implementation, data is aggregated in the form of fragments that are distributed in the network. The default fragment structure is shown in Figure 2.8(a) and is updated periodically as the distributed variable changes its properties. An updated fragment structure is shown in Figure 2.8(b). This technique is compared with existing leader-based, cluster-based and centralized techniques [83][84][85]. Section 2.6 discusses this technique in more detail.

2.2 Related Work

Depending on their abstraction level, existing programming languages for networks of sensors and embedded processors can be grouped into four classes: low level, node level, local neighborhood level, and application level languages [41]. Low level programs explicitly manage the CPU operation, events, and communication between CPUs. Node level languages focus on expressing the shared and local state information of an application, and the operators to process the state through remotely called procedures. Local neighborhood level languages offer constructs to define and manage information across localized clusters. Application level languages express an application's functionality without referring to computing or networking details. A somewhat similar classification is proposed in [87]: programming models are distinguished into node level, group level (neighborhood and logical groups), and network-level (global behavior) models.

Low level languages include nesC, Insense, Query Machine and Mate, among other

languages. nesC language [25], arguably the most popular specification language for sensor networks, extends C language with constructs for supporting events, concurrency, and component-oriented design. Task communication is non-blocking, based on split-phase. Routing uses minimum spanning trees. Insense [55] offers parallel threads communicating through typed channels. Query Machine [53] is a stack-based language with dedicated instructions for sensors, buffers, and data aggregation, in addition to the traditional CPU level instructions, like flow control, arithmetic and logic operations, and stack management. Mate programs [15] are formed out of small code capsules, which self-replicate through the network using a lightweight version of BLISS protocol.

SNACK is a node level language [30]. It includes component composition constructs, and a library of reusable components and services. The language offers support for describing shared and local state, sharing of components with similar parameters, and sharing of control flow or similar actions. Descriptions are based on Remote Procedure Calls of services through well defined interfaces. A similar concepts is implemented in JCells, a language for Internet [16]. In JCells, operators are exported and imported by containers of objects and code through typed interfaces. The containers can be moved, copied, and linked dynamically with each other either locally or over the network. A similar model is proposed in [34, 42].

Local neighborhood level languages provide constructs for accessing data and specifying communications in localized regions [29, 28], such as constructs for data addressing, data sharing, and data reduction. Neighborhood is defined based on metrics like radio connectivity and geographical location. Abstract Regions [29] are algorithmic descriptions that use a data sharing mechanism across neighborhoods of nodes. Programs must define all cluster level operations, like selecting the leader nodes, but the language offers operators for routing data between nodes, aggregating data, and exploring trade offs, e.g., bandwidth vs. energy consumption trade off [58]. Resource brokers are used for fine-grained resource management in Pixie, a dataflow language [60]. Similarly, Hood [28] is a node-centered approach, which extends nesC with constructs for membership, data sharing, caching and messaging. Shared values are updated through policies that use information about neighbors, like state, cached attributes, and local annotations. Logical neighborhoods are discussed in [43].

Application level languages describe the global functionality of an application based on sensor networks without referring to the topology of the network [41, 27,

45, 32]. The functionality is specified as a sequence of operations on continuous data streams, like join, selection, group formation, aggregation, and sort. Extraction of spatial-temporal pattern queries is presented in [31]. Queries are predicates defined over exact or relative time constraints. In Flask [41, 51, 62], applications are described using distributed dataflow graphs, which are compiled into nesC programs. The language abstracts low level aspects, like concurrency, buffering, and routing, but requires explicit description of algorithmic routines and interactions between the routines. Hence, specifications still must include a significant amount of implementation details, like sampling clocks, analog to digital conversions, data flow merging, specific filtering, etc. TelegraphCQ [27] is a dataflow language for adaptive processing of continuous queries. Support for adaptation includes scheduling, resource allocation, and trade offs between flexibility and overhead. The language constructs include interfacing support through ingress and caching, query operators like join, selection, projection, grouping and aggregation, and duplicate elimination, and adaptive routing through eddies to support parallelism, load balancing, and fault tolerance. Each data stream must have an associated state in eddies, such as to indicate the set of modules where a data must be routed to. The Borealis stream processor [32] assumes window-based operators over continuously streaming tuples. The language addresses three important requirements: revision and fault tolerance of the input streams by maintaining replica or generating tentative tuples, dynamic query modification through changing the query attributes at runtime, and dynamic and distributed query optimization by local steps, e.g., priority scheduling, query plan modification, load shedding [52], and global decisions, like query allocation to nodes [33, 46]. MacroLab [59] adds vector programming abstraction to Matlab. Macrovectors are sampled from distributed physical spaces defined by the vectors's scope. Programs describe operations executed over macrovectors, like addition, subtraction, max, min, find, etc.

Declarative programming languages, a particular kind of an application level language, are proposed for data query processing [47, 40], cleaning of unreliable data [39], model identification [47], adaptive sampling and in-network event generation [53]. The Pulse model [47], proposed for continuous-time model identification, describes the mathematical nature of the model and any related constraints, e.g., error, and then uses a gradient-based solver to find the parameters of the model. SwissQM [53] includes declarative constructs for describing window queries, complex event generation, overlapping sensor networks, and optimization. Zhao *et al.* [26, 12] propose Spatial

Aggregation Language (SAL) in which applications are described through creation and transformation of spatial-temporal objects. Object properties, e.g., location, intensity, and motion, emulate the properties of physical objects. The language defines multi-layer hierarchies, i.e. spatial objects, cell complexes, and geometric objects, based on abstraction and composition operators, like aggregate, classify, re-describe, localize, and search. Abstractions describe classes of equivalence between the composing entities. Transformations refer to the properties of Euclidean spaces, and include functions like rotation and translation. [14] presents a kinetic data structure that maintains all the attributes describing objects in continuous motion, such as 2-D convex hull and closest pair.

2.3 Z-based Specification Notation

This work uses a notation based on Z language [24] to define distributed data and events, to declare their properties, and to specify the constraints and goals of the application. This programming notation is at the application level. Similar to other declarative languages [63, 38, 48], it is based on first-order predicate logic to express the functionality of a system with respect to the properties of distributed data and events, and how individual sensor readings are integrated over space and time. In addition, the proposed notation describes other important aspects of CPS, like the application goals and constraints, and predictions of the dynamics of physical data in time and space. The explicit description of goals and constraints is important as CPS involve decision making under time and precision requirements (besides data querying). Moreover, specifying constraints helps efficient resource management by facilitating performance trade-off exploration for optimizing the execution for a variety of requirements, i.e. sensing precision, delay, memory size, communication bandwidth, and power consumption. This notation based on Z language [24] has been used for formal specification and verification of computing systems.

Z descriptions are structured using *schema* notation. A schema can include several predicates defined using universal ($\forall$) and existential ($\exists$) quantifiers over variables of various types, like sets, power sets, Cartesian products, sequences and functions. Predicates are the main means for declaring properties, conditions, and constraints. Every predicate defines the involved variables and clauses, which are constructed using logic operators, like conjunction ($\wedge$), disjunction ($\vee$), implication ($\Rightarrow$), and

equivalence ($\Longleftrightarrow$), and set operators, i.e. union ($\cup$), intersection ($\cap$), difference ($\setminus$), membership ($\in$), and subset ($\subset$).

One schema is defined for each sensor layer of the application. Following is a summary of how Z-based constructs are utilized in specifying the features that are helpful for building high-level specifications:

- *Physically-distributed variables*: Distributed variables are described as continuous functions defined over space and time ranges, such as gas concentration (variable *concentration* in Figure 2.2) and temperature (*temp* in Figure 2.3). Universal and existential operators can be used for such variables.

  For example, the construct $\forall x \in [X_L, X_R].\forall y \in [Y_L, Y_R].\ p(x, y)$ refers to the distributed variable p defined over the points of a planar, rectangular region. $\forall t \in [T_L, T_R].\ q(t)$ refers to the variable q over a time window.

- *Data characteristics*: The sampled data is characterized by a set of basic attributes, including value, precision (e.g., bitwidth and sampling time), position, and time. Each basic attribute can be used to compute aggregated attributes, like the overall value in time and space, and rates of change, such as gradients in time and space. This is expressed through integral operator $\int$ and differential operator ∂, respectively.

- *First-order predicate logic*: Predicates are constructed using operators $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Longleftrightarrow$, $\forall$, and $\exists$. The attribute *response* indicates the time moment of completing the operator evaluation for distributed variables. The consequence of an operator $\Rightarrow$ is performed if its premise is true. In addition to declaring properties and constraints, the operator also specifies reactive behavior, in which an event triggers a response. For example, in Figure 2.3, temperature sensing generates response $Action_1$, if the precision of the reading falls below threshold Lim_{prec}. The attribute *response* of method $Action_1$ represents the response time, and is used to describe timing constraints.

- *Set operators*: As some distributed variables are sets and sequences, the following set-related operators are available: $\subset$, $\subseteq$, $\cup$, $\cap$, $\in$, $\setminus$, and P for defining the power set. Constructs $\{Signature|Predicate.Expression\}$ define implicit sets where $Signature$ specifies the domain of the variables, $Predicate$ is the filter for

$\mathbf{GasType} : \{\mathbf{No_x}, \mathbf{CO}, \mathbf{CO_2}, \mathbf{SO_2}\}$
$\mathbf{ExpConcentrationType} : \mathbf{GasType} \times \mathbf{DOM_x} \times \mathbf{DOM_y} \times \mathbf{Time} \rightarrow \mathbf{R_+}$
$\mathbf{TimeI} : [\mathbf{T_L}, \mathbf{T_R}], \mathbf{T_L}, \mathbf{T_R} \in \mathbf{R_+}$ and $\mathbf{T_L} < \mathbf{T_R}$

Application[AltPaths, Start, End]

$\mathbf{TotalPollution!} : \text{seq}\, \mathbf{ExpConcentrationType}$
$\mathbf{Delay!} : \text{seq}\ \mathbf{TimeI}$
$\mathbf{ExpConcentration?} : \mathbf{ExpConcentrationType}$

$\forall \mathbf{gas} \in \mathbf{GasType}. \forall \mathbf{path} : \mathbf{AltPaths}. \forall \mathbf{t} : \mathbf{Time}. \forall (\mathbf{x}, \mathbf{y}) \in \mathbf{path} :$
$\mathbf{TotalPollution!} = \mathbf{TotalPollution} \cup \mathbf{ExpConcentration?}(\mathbf{gas}, \mathbf{x}, \mathbf{y}, \mathbf{t})$
$\exists \mathbf{t} : \mathbf{Time}. \forall \mathbf{e} \in \mathbf{TotalPollution}.$
$(\mathbf{t} < \mathbf{t_0}) \wedge (\mathbf{t} = \max \mathbf{e.Time}) \wedge (\mathbf{t_0} - \mathbf{t} > \mathbf{Lim_{req}}) \Rightarrow (\mathbf{Delay!} = \mathbf{Delay} \cup [\mathbf{t}, \mathbf{t_0}])$
$\exists \mathbf{path} \in \mathbf{AltPaths}. \min(\alpha \int_{\mathbf{path}} \int_{\mathbf{Start}}^{\mathbf{End}} \mathbf{TotalPollution}(\mathbf{Type}, \mathbf{l}, \mathbf{t}) \mathbf{dtdl} + \beta \sum \mathbf{Delay})$

Figure 2.1: Schema for describing the application goals

selecting the values that make the predicate valid, and *Expression* represents the computation that produces the values in the set.

- *Invariant constraints*: Invariants, used in prediction, are predicates over distributed variables to define properties valid for any physical point and/or time instance. The predicates refer to time instances following the current moment t_0 and to physical points that are different from the node performing the prediction. As they can include derivative and difference operators, invariants can also describe the dynamics of distributed variables, i.e. variations along curves and over time.

- *Goals and constraints*: Goals describe the expressions to be maximized or minimized by decision making. Goals are specified using integral or differential operators defined over distributed variables. Constraints are predicates that represent requirements for data attributes. Timing constraints use attribute *response* of an operator over distributed data. The attribute denotes the time moment when the operator completes.

The next section discusses the Z descriptions of the three sensor layers of the application described in the illustrative example in Figure 1.1.

2.4 Z-based Specification of Illustrating Example

Schema *Application* in Figure 2.1 describes the application goals. Constants DOM_x and DOM_y are the ranges of coordinates x and y of the monitored area. The generic parameters define the set of alternative paths ($AltPaths$) analyzed by the decision

making routine in Figure 1.1 to find the path of minimum total pollution, and start time (*Start*) and end time (*End*) of the time window considered for analysis. Output variable *TotalPollution* (Z output names end in '!') is used in computing the total pollution of a path. The variable is a sequence of values *ExpConcentration*, where each value defines the pollution level (current and future) of a path segment.

ExpConcentration is an input (schema input names end in '?'), and is produced by schema *GasCloud* in Figure 2.2. Output *Delay* is a sequence of time ranges. Each range is a time interval during which the sensing precision was below the needed resolution.

The bottom part of the schema defines two predicates and the goal of the application. The first predicate indicates the expected pollution of a path: at any time moment, *TotalPollution* includes the current and expected pollution levels of all points forming the path. The second predicate states that a new time interval is added to output *Delay*, if there exists a time moment t prior to the current time t_0, and there exists an element e in sequence *TotalPollution*, such that e does not include a pollution value later than t and the difference between t_0 and t exceed the acceptable limit Lim_{req}. The predicate states that the pollution data available for that segment is obsolete as the time separation between the decision making moment (current time t_0) and the data sensing time (t) is too large. Lim_{req} is a constant. The third equation defines the application goal, which is to select the path that minimizes the total pollution (by integrating together the pollution values of the path segments based on the sequence *TotalPollution*), and minimizes the total time over which the required sampling precision is not met (to maintain the validity of predictions). α and β are fixed by the user based on the importance of the two factors.

Schema *GasCloud* in Figure 2.2 specifies gas level monitoring. It defines the conditions under which a new gas cloud emerges, the data collected about cloud position, size, and trajectory, and predictions about the gas levels at other physical positions and future time instances. To simplify the description, only the earliest-detected gas cloud is monitored, but this does not limit the concept. To avoid false alarms, constant Lim_{gas} is the threshold above which a gas has been detected in a point, and constants Lim_x and Lim_y are thresholds defining the minimum size of a cloud. Input *concentration* returns the instantaneous concentration level of a gas at a sensing point, e.g., using an API method of the gas sensor. Output *Type* stores the gas type of a cloud. Output T indicates the sequence of time ranges during

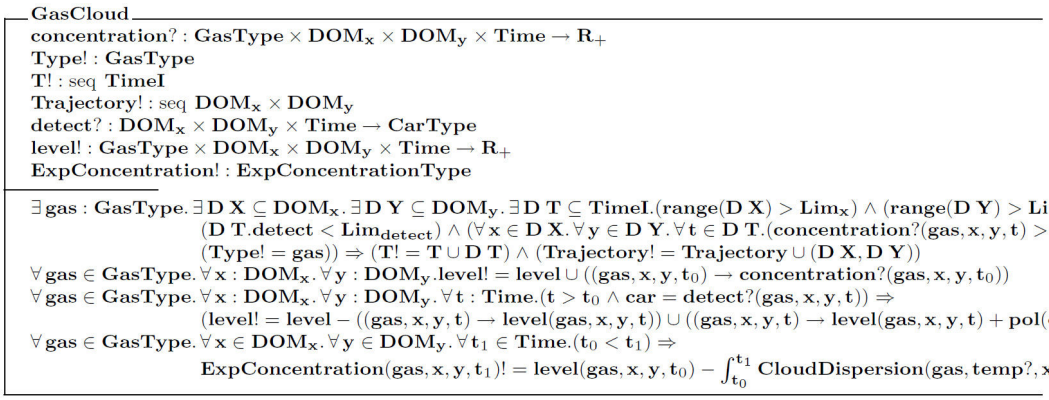

Figure 2.2: Schema for describing gas cloud detection and gas pollution concentration monitoring

which the gas cloud was detected, and which form the lifetime of the cloud. Output *Trajectory* is the sequence of positions forming the trajectory of the cloud. Input *detect* indicates if a new polluting object (e.g., a car) was detected in a point of the physical space at time t, such as using sound-based tracking [64]. Output *level* describes the pollution level at a point and time. It is updated based on the gas sensor readings and upon detecting a new vehicle. Output *ExpConcentration* expresses the current and expected pollution levels of any point of the monitored zone.

The bottom part of the schema has four predicates. The first predicate states that a new gas cloud emerges if there exists a rectangular, compact region over which the toxic gas concentration at any point exceeds threshold Lim_{gas}. The size of the detected region is defined by ranges $D\ X$ and $D\ Y$. To avoid false alarms, the dimensions of the cloud along the two coordinates must be larger than thresholds Lim_x and Lim_y. The constraint must be satisfied for the entire lifetime of the cloud. The predicate records the type of the detected gas cloud in the output variable $Type$, the time instances at which the cloud exists in output sequence T, and the trajectory of the cloud in output sequence $Trajectory$. The second predicate states that in any point the pollution level at the current moment t_0 is equal to the reading of the gas sensor. The third predicate expresses the invariant rule used to predict function *level* at time moments after the current time. If a car is detected, the level in that point increases by the typical amount of pollution created by that car. Function *pol* is a car's typical pollution, which is built through experiments. A different schema can be constructed that would characterize this function *pol* that depends on different conditions. The

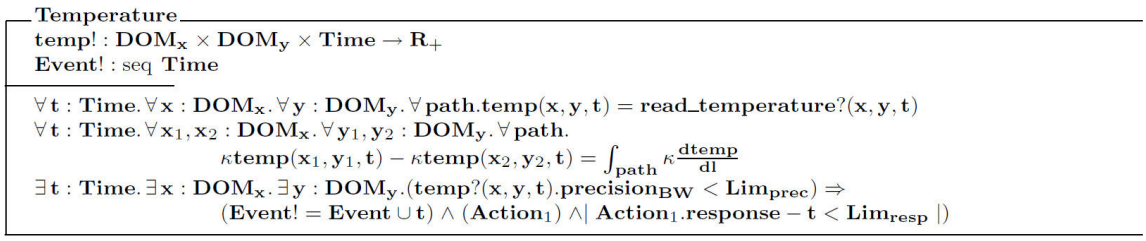

Figure 2.3: Schema for describing temperature sensing

predicates can be specified such that data from acoustic sensors can estimate the speed of the car and its relative position with respect to other cars. This data can be used to extract driver profile. Another predicate can characterize the type of car (sedan, SUVs, trucks, etc.). This information can be obtained through classification techniques by using the frequency signature of acoustic data. Combining information extracted from these two predicates can be used to estimate the pollution level of the car. The fourth predicate of Schema *GasCloud* is an invariant equation that defines the expected gas concentration at a future time t_1 depending on the concentration level at time t_1 (considering predictions about vehicles traversing the point) and the amount of dispersed gas between t_0 and t_1. The invariant expresses the mass conservation law for gases [13]. Note that function *CloudDispersion*, that captures the gas dispersion, has input parameter *temp*, which introduces a dependency on the temperature sensing layer.

Schema *Temperature* in Figure 2.3 describes the temperature sensing layer. Constant κ is the heat transfer coefficient. Constants Lim_{req} and Lim_{prec} are threshold values defining the minimum precision requirements for temperature sampling. Lim_{resp} is the timing constraint for response to temperature related events. The schema variables include output *temp* that stores the temperature reading at any point and time. Output *Event* is a sequence recording the events associated to temperature sensing.

The first predicate states that *temp* is equal to the temperature sensor reading at that point, e.g., using the sensor's API method *read_temperature*. The second predicate defines the invariant characteristic of temperature sensing, which states that the temperature values in any two points x_1, y_1 and x_2, y_2 are correlated by the integral of the heat transfer coefficient along any path between the two points [13]. The third predicate records in output *Event* all time events at which the temperature

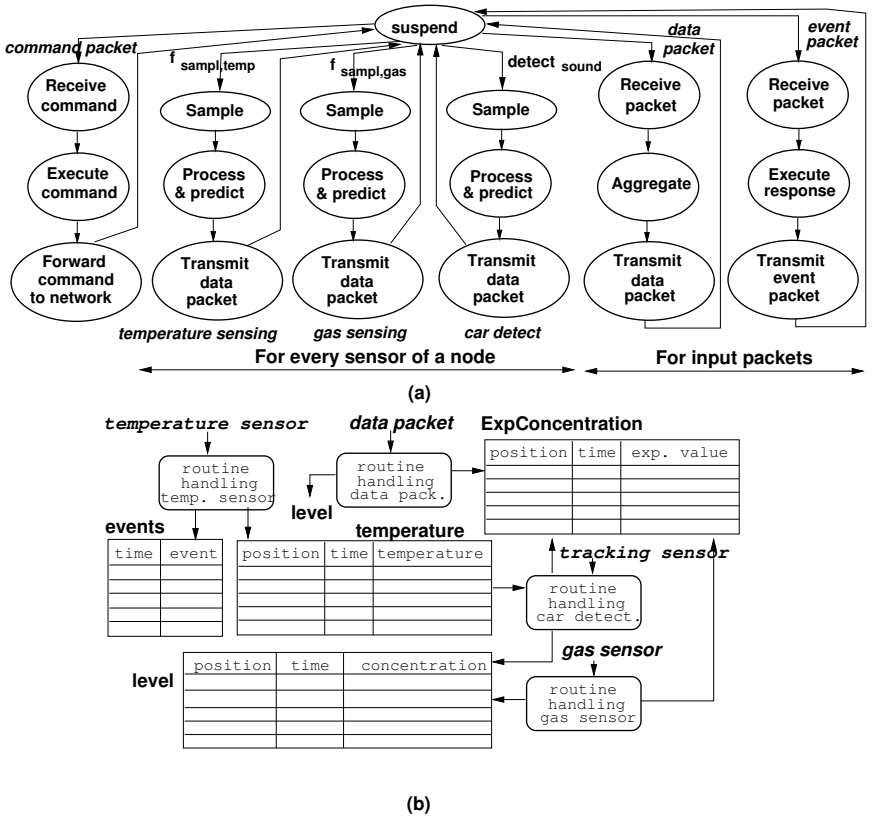

Figure 2.4: Code and data structures of the reconfigurable sensor nodes

sensing precision had fallen bellow limit Lim_{prec}. Attribute $precision_{BW}$ of $temp$ indicates the bitwidth precision of sensing. For every event, the predefined action $Action_1$ is performed under the timing constraint that its response time must be less than constant Lim_{resp}.

2.5 Translation of Z-based descriptions into executable code

The methodology for translating Z-based descriptions into executable code is summarized next. Each sensing device (identified by its input APIs) has an execution thread, as shown in Figure 2.4(a). The code for the three steps of a thread, (i) sampling, (ii) process and predict, and (iii) transmit data packet, is created using the predicates that involve only variables related to a single physical position (i.e. $read\_temperature?(x, y, t)$ which refers to the physical position of coordinates x and

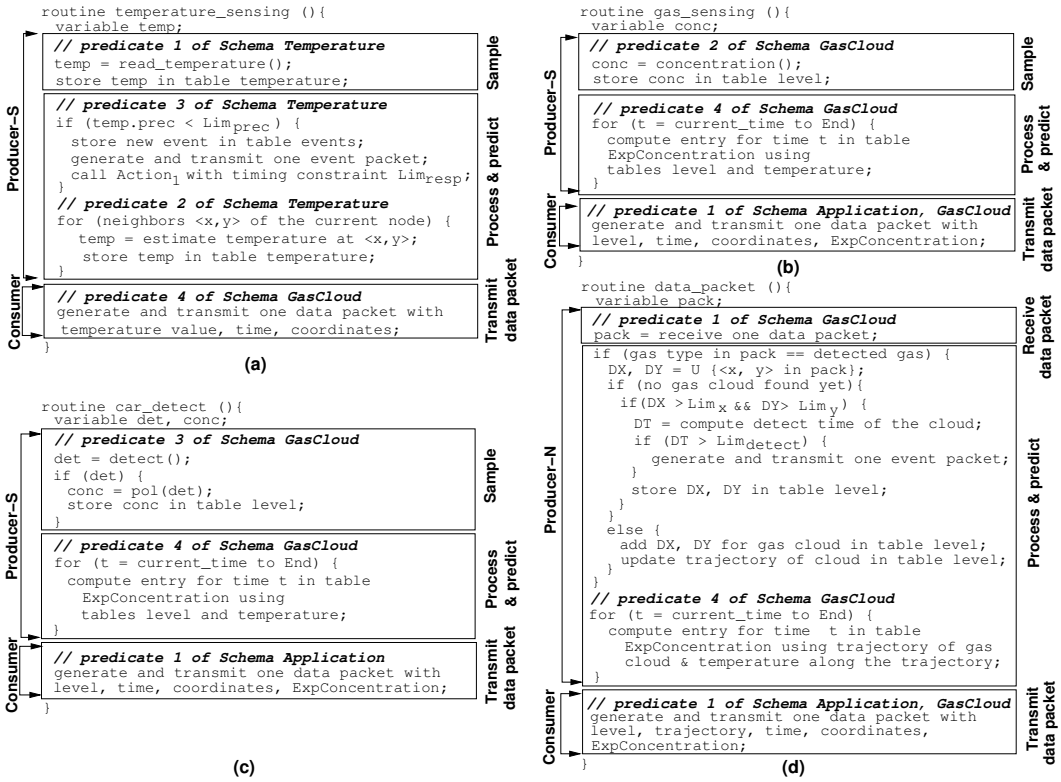

Figure 2.5: Sensor node code that implements the example in Section 3

y) and not a broader physical area (e.g., variable $Trajectory$ that refers to the coordinate ranges DX and DY). The predicates involving variables defined over physical areas produce code for the threads corresponding to data and event packets. The sequencing of the code of the three steps (of a thread) is not important as every Z rule defines a situation that does not conflict with the other rules. Logic implications produce if statements, in which the premise is the evaluated condition and the conclusion represents the body of the statement. Data tables, as shown in Figure 2.4(b), correspond to the variables and functions of the Z-based specification. Instructions referring to the equality operator produce code to update the data tables.

The routine in Figure 2.5(a) implements the SN functionality for sampling the temperature sensors. It corresponds to the second thread in Figure 2.4(a), and is executed with frequency $f_{sampl,temp}$, which is a parameter of the architecture. As expressed by the first predicate of Schema $Temperature$ in Figure 2.3, the sensor inputs are sampled using the sensor's API $read\_temperature()$. The values are stored in the local table *temperature* of a node, including the position, time, and value of the sampled temperature. The table structure corresponds to variable *temp* in Figure 2.3.

As declared by the third predicate of Schema *Temperature*, if the bitwidth precision of sampling is less than Lim_{prec}, an event is generated: the event is stored locally in Table *events*, e.g., the event time and identifier. An event packet is produced to notify the target point. In addition, the node executes response $Action_1$ with timing constraint Lim_{resp}. Then, according to the second predicate of Schema *Temperature*, the temperature values at physical points different than the sensor node position are estimated and stored in the local table. The estimation uses the physical law for temperature variation describing the temperature difference along a path. Finally, a data packet is created and transmitted to the target point with information about the sampled temperature. The data packet is used to implement predicate four of Schema *GasCloud*.

Figure 2.5(b) describes the routine handling gas sensors (third thread in Figure 2.4(a)). The routine uses tables *level* and *ExpConcentration* in Figure 2.4(b). Table *level* keeps gas-related information at a point, i.e. coordinates, time, and gas concentration level. Table *ExpConcentration* is used to predict the gas concentration at future time moments. The structures of the two tables correspond to the corresponding variable descriptions in Figures 2.1 and 2.2. The routine is executed with frequency $f_{sampl,gas}$. First, it samples the gas using the sensor's API *concentration*(), and then stores in table *level* the sampled information. These actions are according to the second predicate of schema *GasCloud* in Figure 2.2. Then, as declared by the fourth predicate of schema *GasCloud*, the expected gas level concentration is computed for the time window defined by the current time moment and constant *End* provided by the decision making routine. The expected level is found based on the physical law of gas dispersion. A data packet is then formed with gas information and sent to the target point, as required by the first predicates of Schema *Application* and *GasCloud*. The two tables are also updated, if a car is detected by the tracking sensor, as expressed by the third predicate of schema *GasCloud*. *pol*() indicates the typical pollution level of a certain car type denoted by variable *det*. The code in Figure 2.5(c) is executed by the routine handling inputs from the SN's sonar used in tracking vehicles (fourth thread in Figure 2.4(a)). Upon detecting a moving vehicle through the sonar's API *detect*(), the routine updates table *level* by adding the extra pollution levels due to the car, as defined by the third predicate of schema *GasCloud*, and then also updates table *ExpConcentration*. Note that computing the predicted values utilizes table *temperature*, as expressed by the third predicate. This justifies

the transmission of the temperature values sensed by other nodes.

Figure 2.5(d) presents the code used to handle the received data packets that include information about gases. The figure corresponds to the fifth thread in Figure 2.4(a). According to the first predicate of schema *GasCloud*, the received gas-related information is used to detect emergent gas clouds by clustering the positions of the neighboring points at which the gas was detected. Variables x and y (in a packet) describe points in space at which the gas was detected. The size of a cloud is indicated by variables DX and DY. If the coordinates of the clusters exceed limits Lim_x and Lim_y, the time for detecting the cloud, DT, and its trajectory are stored. If the detection time exceeds the threshold Lim_{detect}, an event packet is transmitted to notify the target point indicating that the delay of transmitting data packets exceeds the timing constraint. Table *level* is updated with information about the gas cloud. Then, as expressed in predicate four of Schema *GasCloud*, table *ExpConcentration* is updated with information about the gas cloud using the gas dispersion law, and the concentration level and temperature along paths. Finally, data packets are created and transmitted to the target point.

Event packets received by a node are forwarded to the target point.

Table *ExpConcentration* is used by the target point to evaluate the total pollution of the analyzed paths (first predicate of schema *Application* in Figure 2.1) and to find the time range for which data acquisition precision is not sufficient (second predicate of the schema).

2.6 Representing Distributed Variables using Kinetic Data Structures

As discussed in the previous sections, the distributed data that corresponds to an entity being monitored is considered a distributed variable. These distributed variables are represented using a procedure based on Kinetic Data Structures (KDS) [86] since they keep changing their properties in time and space, i.e. they are kinetic in nature. The procedure uses the concept of KDS to implement a decentralized methodology where data is aggregated in the form of fragments distributed over the entire cloud. When an emergent cloud is detected, we compute the initial set of fragments in the form of convex hulls as shown in Figure 2.8(a). Each fragment is represented by an

aggregate node that correlates its data with other aggregate nodes to make decisions that correspond to merging or splitting those fragments to represent a phenomenon. Whenever these fragments change their structure, related parameters, like area, location, density and composition, are updated.

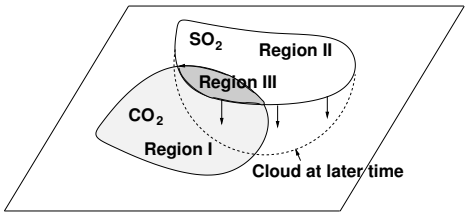

Figure 2.6: KDS for geographically distributed, physical data

Accurate data representation is directly related to adjusting network granularity depending on data distribution over space, computing the optimal value of sampling precision, and minimizing data loss and latency. The procedure to adjust network granularity is performed by merging or splitting KDS fragments depending on information gradients over distances. It is also important that the local regions do not lose their identities when entities overlap. For example in Figure 2.6, regions I and II should be correctly represented even when an overlapping region III is formed. Updates over time should not lose information. When region II in Figure 2.6 expands over time, fragments inside the network need to be updated in a time-efficient manner. The subsection on *Comparison with existing approaches* explains why existing leader-based approaches are insufficient and the advantages of using the distributed KDS-based approach over existing leader-based approaches are experimentally proven.

The process of obtaining a meaningful representation of the cloud using the KDS-based method is summarized in Figure 2.7. At the lowest level, the sensor nodes sample emergent events and compute the attribute vectors (AV). The vectors are then used to compute the initial set of convex hulls. Data is aggregated at this level, so that each Aggregation Node (AN) contains the aggregated attribute vector (AAV) for that particular convex hull. This aggregated data can be used to compute several topographical parameters such as boundary, area and location of the convex hull.

2.6.1 KDS Parameters

In order to faithfully represent the dynamic nature of the physical cloud movement, we use KDS to characterize different aspects of the entities which form the cloud.

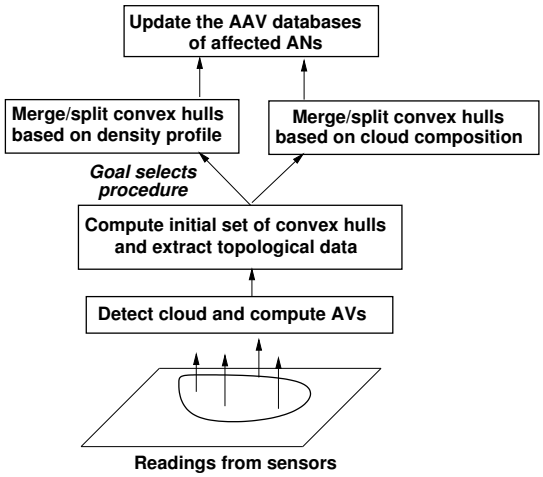

Figure 2.7: Proposed flow to construct distributed KDS

Our cloud KDS have three main parameters:

- *Topography*: The parameter describes the geographical extent of the cloud. It can express boundary, area, and location. In order to extract these parameters, the sensing nodes generate attributes by providing either the location of the entity, or their own location if they do not have localization capabilities.

- *Composition*: The parameter contains information regarding the inherent signature characteristics of the entity being monitored. For example, the chemical composition of a gas or the frequency spectrum of a sound source. The parameter helps us to define another parameter called cloud class, e.g., homogeneous or heterogeneous. The attributes required for this purpose are the signature attributes, which are generated by the nodes using their sampled data.

- *Density*: The parameter contains information regarding the density of the entity in different regions of the cloud. Combining the data from all these parameters, we can identify important derived characteristics such as the continuous/discontinuous nature of the cloud, merging/splitting of clouds, and movement of clouds over time.

2.6.2 KDS Operators

KDS include three main operators for each kind of parameter: (i) convex hull operator [86] is the mathematical model that supports aggregating data fragments into

Algorithm 1 Algorithm for computing the initial convex hulls

```
for each active node i = 1 to n do
  initialize count = 0
  forward data towards the ANs
  if (y_i ≠ 0)and(x_i ≠ N) then
    collect incoming data vectors to compute the convex hull at AN_i
    increment count
    if count > 2 then
      aggregate data in the format: AAV < (X,Y)_A, H_A, D_A >
    end if
  end if
end for
```

a valid result, (ii) composition operator that decides to merge fragments (applying convex hull operator) depending on the pursued goal, and (iii) splitting operator that decomposes the computing of a KDS parameter for a larger geographical area into fragments corresponding to smaller regions. The three operators are presented next.

i. Convex hulls: Suppose an emergent cloud triggered events at n nodes in a network of size $N \times N$. The n nodes sample the data and compute the low level attributes that are required to comprehensively characterize the KDS operators. These attributes are then converted into aggregated fragments called convex hulls (CH) at the Aggregation Nodes (AN), using certain aggregation scheme as shown in Figure 2.8(a). The arrows show the communication scheme for transmitting data during the aggregation process. The procedure used to form these initial set of convex hulls is explained below. The input to this function is the attribute vector generated by the sampling nodes, represented as follows:

$$AV_i < (x, y)_i, H_i, D_i > \tag{2.1}$$

$$H_i < entity_1, entity_2, ..entity_l > \tag{2.2}$$

$$D_i < d_1, d_2, ..d_l > \tag{2.3}$$

where, (x, y) is the location of the sensor node, H is the cumulative vector which contains the list of l entities, D contains the density of each type of entity, in the

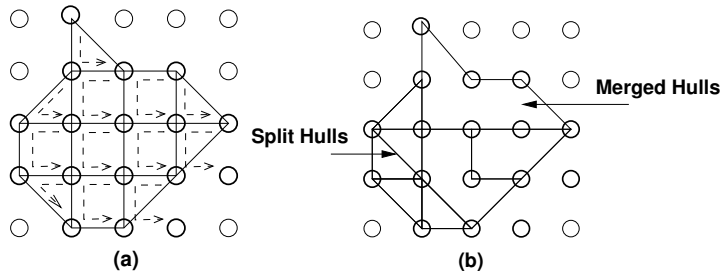

Figure 2.8: Aggregation, splitting and merging of convex hull fragments

same order as the names in H. Each node generates one attribute vector. So, i varies from 1 to n.

In Algorithm 1, the initial convex hulls are formed using a static algorithm where the bottom-right node in the structure is the AN for that particular convex hull. Therefore, the nodes which have y coordinate zero (leftmost column) or those which have x coordinate equal to N (topmost row), cannot be ANs. Also, the AN requires at least three active nodes to form a convex hull. The aggregated attribute vector contains $(X, Y)_A$ which is a list of (x, y) coordinates of the nodes which form the convex hull, H_A is the cumulative aggregated vector of the list of distinct entities and D_A contains the average density of the respective entities for that particular convex hull.

Algorithm 2 Goal-Oriented Merging/Splitting of Convex hulls - Density profile

```
send AAVs to neighboring active ANs
receive AAVs from neighboring active ANs
if goal = Density profile then
  for each AN j = 1 to m do
    compare density profile D for entities
    if dD/d(x,y) > threshold_high then
      split convex hull into smaller sections
    else
      if dD/d(x,y) < threshold_low then
        merge convex hulls into larger sections
      end if
    end if
  end for
end if
Update convex hull information in AAV databases in the ANs
```

Algorithm 3 Goal-Oriented Merging/Splitting of Convex hulls - Cloud composition

```
send AAVs to neighboring active ANs
receive AAVs from neighboring active ANs
if goal = Cloud composition then
  for each AN j = 1 to m do
    compare list of entities H_A in the incoming AAVs
    if lists match then
      combine hulls
    end if
  end for
end if
Update convex hull information in AAV databases in the ANs
```

ii-iii. Goal-oriented composition (merging) and distribution (splitting): In the analysis, n nodes form m aggregation nodes, which in turn compute p convex hulls. Initially, the ANs transmit their AAVs to the active neighbors and receive AAVs from others. Depending on the goal, Algorithm 2 or Algorithm 3 is selected to operate on the network. Therefore, depending on the density gradient or composition of the cloud, convex hulls are split (distributed) or merged (composed), to get more insights into the cloud characteristics as shown in Figure 2.8(b).

2.6.3 Comparison with existing approaches

Existing leader-based approaches [87][83][84][85] have data aggregated onto leader nodes and the leader nodes process their received data to compute the regions of interest and represent the global structure of a geographically-distributed data, e.g., a data cloud. In hierarchical leader-based approaches [82], there are local leaders that compute regions that get updated at the root node. These approaches are more efficient compared to the traditional approach where individual data is sent to a base station and the base station performs all computations. They are well suited for query-based applications.

However, unlike query-based applications, CPS applications involves the accurate representation of a distributed variable. These entities change their properties in space and time and there is a need for a data structure that would dynamically evolve as these properties change. This data structure describes the entity in the form of a set of variables, that include Topography, Composition and Density of the entity. Even

though existing leader-based approaches reduce communication overhead compared to the traditional approach, large amount of data still has to be communicated to the leader node or the root node. The leader node then processes the received data and if necessary, transmits decisions back to the nodes local to monitored entity if the application is not only query-based. However, the KDS-based methodology we use is decentralized and distributed among the nodes that sample the entity, thus further reducing communication overhead significantly compared to the leader based approach. Another advantage of this technique over leader based approach is that based on the granularity of the local fragment in which a node lies with respect to a specific property (e.g. density, composition, etc.), the node can derive information regarding the dynamics of the entity and can make its own decisions locally without the need for a leader node to process data and transmit decisions to the node. The granularity of a fragment is the level at which it is clustered with respect to other fragments. For example, compared to the default fragment structure in Figure 2.8(a), the merged fragments in Figure 2.8(b) would represent lower density gradients and the split fragments would represent higher density gradients of the gas cloud. Hence, nodes that are local to these fragments would have sufficient information regarding the gas cloud to make any local decisions. Hence, we can conclude from the above mentioned advantages that by locally updating the fragments based on the changing properties of the entity, the KDS-based distributed technique is quicker to react to the dynamic nature of the distributed variables.

To prove that the KDS-based technique reduces communication overhead significantly compared to the existing techniques, the performance of the distributed algorithm was studied on three data points or cloud structures that were defined based on their dynamics. Experiments were conducted on a 100 node PSoC Network simulator that functions using the execution support described in Chapter 3. Once the initial fragmented structure (convex hulls) are formed when the cloud emerges and the fragments are merged or split depending on information gradients, it should be less expensive in terms of latency to update the kinetic data structure operators. In this approach, existing operator values are updated without having to recompute all parameters. When an event emerges in the network, it is incorporated into its closest fragments and all operators related to its topography, composition and density are updated in a time-efficient manner.

Every cloud structure defined is updated in five iterations after the initial frag-

Parameters	Cloud Structure based on dynamics	Traditional Method	Cluster-head based method	Hierarchical-based method [82]
Percentage	Slow	89	48	25
Latency	Fast	91	69	25
Improvement	Merging	88	74	25
Percentage	Slow	82	73	60
Data Loss	Fast	81	64	40
Improvement	Merging	68	46	0

Table 2.1: Percentage improvements over latency and data loss of the proposed technique

ments representing its properties are formed. Each data cloud has fifteen to twenty events active during a particular time. The number of events correspond to the number of nodes sampling relevant data.

The first data cloud structure moves slowly over the network and due to its slow dynamics, the network is efficiently able to update its fragmented structures as data clouds move. Only 18-20% of the whole fragmented structure has to be updated on an average as the cloud moves.

The second data cloud structure moves at a rate faster than the processing capability of the network and hence, the network sees the change in dynamics to be abrupt. 35-45% of the whole fragmented structure has to be updated on an average as this cloud moves over the network.

The third data cloud structure starts with two separate clouds, each with eight and nine events respectively. Eventually they merge to form a single cloud with 17 events. The region common between those clouds experiences a sudden surge in densities. As a results, smaller fragments of convex hulls were formed by using our splitting procedure to obtain more information in those areas of high density gradients. As a result, the nodes in those regions experienced higher data loss values.

Since the fragmented approach is used which involves only local data communication, the latency value is consistently similar for all types of cloud structures. We could observe only a difference of only up to 10% in latency values for the same resource utility over different cloud structures and it is fairly consistent at 60 to 70 msec. when default resource parameters are used. As expected, the slow cloud produced the lowest data loss of 4 compared to the other clouds, the fast cloud produced a data loss of 6 and the merging cloud produced the highest data loss of 7.

The described technique is compared to existing approaches in Table 2.1. The traditional approach is the centralized approach where raw data is transmitted to a base station. It can be observed that latency improvements are in the range of 90% and data loss improvement is 75% on an average. Compared to the cluster based approach, we improve latency by an average of 64% and data loss by 61%. The hierarchical based approach also produces fairly consistent values of latency due to its fixed hierarchical structure where local leaders collaborate and update leaders at the higher hierarchy level. This approach is the closest to our distributed approach and we improve latency by 25% and improvements in data loss depends on the cloud structure.

Hence, due to the fact that fragments are updated locally and communication overhead is reduced significantly compared to existing approaches, the KDS-based distributed technique is quick to react to the dynamic nature of the distributed variables making it suitable to represent distributed variables.

2.7 Conclusions

This chapter presents a novel goal-oriented programming framework for developing CPS applications executed on physically-distributed networks of reconfigurable embedded nodes. The considered applications involve physically distributed data and events, objects emerging dynamically, and data requiring predictions about their evolution in space and time. The framework is based on a declarative, Z-based programming notation, and descriptions express the optimization goals, sensed inputs, actuation outputs, events, and performance constraints, while leaving to the optimizer and execution environment the task of optimally implementing the functionality. Goals and constraints are expressed based on distributed variables that specify data acquired over distributed physical regions. The operators performed on distributed variables include set-related operators, logic operators, and cumulative operators, like integrals and differentials. The method to convert high-level description to executable code is also discussed.

The distributed variables, that correspond to the monitored entity, are represented using a distributed technique that uses the concept of Kinetic Data Structures (KDS) where data is aggregated in the form of fragments and gradients within the entity are described either by merging or splitting these fragments, thus changing its gran-

ularity. Information regarding the entities is stored locally and only a subset of these fragments have to be updated with the changing properties of the cloud (35%-45% of the structure for fast moving clouds). In addition, latency improvement ranging from 25% to 90% compared to existing methods makes this technique capable of quickly adapting to any changes in the properties of the distributed variable.

Chapter 3

Execution Support of the Networked Embedded Infrastructure[1]

3.1 Introduction

This chapter provides the execution platform that defines middleware routines to run network-level applications on the networked reconfigurable sensored system. Each sensor node is a PSoC embedded processor [37] and these nodes are interconnected using UART modules and/or radio communication modules. The middleware routines generate either command, data or event packets. The command packets specify network parameters like regions, data paths, target points, sampling rates, sampling precision, event threshold values and actuation procedures. The values of these parameters are generated based on the design points and executable code that are generated using the goal-oriented specifications discussed in Chapter 2. In the experiments section, this chapter also provides timing and power analysis of these middleware routines and network performance related measurements for a grid network of reconfigurable embedded PSoC processors.

This chapter also discusses a novel method to detect emergent physical entities, e.g., clouds of polluting gas, or clusters of autonomous agents, like robots or vehicles using a decentralized technique. Emergent entities in CPS express phenomena that

[1]The work on detecting emergent entities was published in [5]

are distributed in space and time, and are dynamic with respect to their characteristic attributes and lifetime. Due to the dynamic nature of these distributed entities, there is a need for timely detection of the emergence of these entities and an efficient procedure to accurately represent these entities using the process of building data models. This chapter focuses on the part of timely detection of these emergent phenomena, which corresponds to a distributed event.

The research community defines events simply as deviating sampled measurements that violate a pre-defined threshold condition. [88] presents an event-driven wireless visual sensor network where surveillance frames are transmitted to cluster-heads only when an event is detected using a threshold condition. [89] conducts a thorough survey of existing outlier detection techniques for wireless sensor networks. Outliers are measurements that significantly deviate from normal pattern of sensed data. Reasons for outliers include noise, sensing errors, events and external attacks on the network. [90] presents a method for event detection where nodes are initially trained by exposing them to various events for feature extraction and a control station extracts events from these features.

During the process of detecting emergent entities, events only at the node level are considered as measurements that violate a threshold condition. For example, a node would produce an event for the application example described in Figure 1.1 if it exceeds the measurement from the gas sensor exceeds the density threshold Lim_{gas} defined in the schema describing gas cloud detection in Figure 2.2.

At the network-level, we define events as measurements that correspond to emergent data and these events are used to reliably detect new entities that appear in the network vicinity. These events, that are distributed among nodes, are generated in the form of asynchronous inputs and they also need to satisfy the parametric constraints set by the application. In the application example, the schema in Figure 2.2 describes these parametric constraints in terms of minimum density of the gas cloud Lim_{gas} to avoid false alarms, and the minimum area covered by the cloud is defined using the variables Lim_x and Lim_y. The nodes in the network are also not assumed to be synchronized with a common clock. Current event detection techniques cannot address these challenges creating a need to devise a new event detection technique. Once an emergent entity is detected and the initial data model is built to represent the entity, its change in dynamics and structure can be tracked in a time-efficient manner through inter-node collaborations and regularly updating the data model parameters.

This procedure of detecting an emergent entity is considered a capability of the execution support in addition to the middleware routines. Using these capabilities of the execution support, experiments were conducted for different algorithms and adaptation policies that were developed to achieve the primary objective of this thesis. These experiments were conducted either on the physical PSoC grid network, or on an efficient SystemC simulator that closely emulates the physical network.

3.2 Execution Support

The execution platform is a grid network of reconfigurable embedded sensor nodes (SNs).

Figure 2.4(a) describes the execution flow of a SN. Each node executes the same functional template based on four kinds of inputs [61]:

- *Command packets* are sent at start-up by the server to the SNs to program the execution parameters of the routines. This procedure transmits the executable code and design points generated based on the high-level specifications to the individual nodes. All nodes execute the same code. The packets implement the following commands: they specify the x and y coordinates of the sensor nodes. Regions within the network are defined by specifying the x and y coordinates of the bottom-left and top-right corners of a rectangular, physical region. These packets also define the common collection point called as target point, the region's paths used in transmitting data to the target points, the paths' transmission parameters, the regions' sensing precisions (i.e. bitwidth resolutions and time intervals between successive measurements), and the regions' aggregation function. After executing the command, the node forwards the packet to its neighbors until all SNs get the packet.

- *Sensed data* correspond to the various sensing layers of the application. Each sensing layer has a separate thread, which is activated depending on the sampling requirement of the application. Each thread includes three steps: acquiring the data from the sensor using the sensor's API methods, processing the data and computing predictions required for decision making, and producing and transmitting the data packets necessary to aggregate distributed data and their properties.

- *Data packets* transmit the sampled and processed data and locally computed model parameters between SNs and the target point to accumulate distributed data for the monitored region. After receiving the packet, a node integrates any additive attributes for which the individual values are not required at the target point for decision making. Then, the packet is forwarded towards the target point.

- *Event packets* communicate events over the network. In response to an event packet, a node might execute an action and then forward the packet to the target point. Events at the node level occur when the sampled data violates a threshold value after being processed by the sensor node.

A more detailed description of the above packets can be found in Appendix A. Execution follows the scheme in Figure 2.4(a), and utilizes the data structures in Figure 2.4(b).

3.3 Event detection using distributed interrupts

This section describes a technique that addresses the challenges of detecting emergent entities. The term distributed interrupts is used to describe this procedure due to the asynchronous and distributed nature of these events. In addition to detecting these emergent entities, there is a need to compute the time taken by the network to detect an emergent entity after it was produced. We define this time period as distributed interrupt latency. Accurately predicting interrupt latency is a challenging task since nodes in the network are not synchronized by a common clock. Accurate prediction of interrupt latency is critical since it is used to compute the criticality of processing an interrupt to generate interrupt priorities and also set application-specific timing constraints for interrupt processing and decision making modules.

The algorithm uses the concept of fragments which was discussed in Chapter 2 where each fragment is represented by an aggregation node (AN). For simplicity, a fragment in this algorithm is considered to have four nodes. A bottom-up three level hierarchical structure shown in figure 3.1 is used by this algorithm to detect emergent entities. The first level of hierarchy is called the node level that detects an emergent entity from sensor notifications. The second level of hierarchy is called the intra-fragment level, where the algorithm performs intra-fragment communications and the

notification packets are transmitted to the respective aggregation nodes (ANs). After forming emergent events, the ANs determine if they are potential interrupts (PI). At the highest level of hierarchy called the inter-fragment level, the ANs perform inter-fragment communications and collaborate with each other to decide if PIs can form an emergent event resulting in interrupt generation. The AN that detects an interrupt broadcasts an interrupt generation packet.

The algorithm computes interrupt latency and the equations to compute it are governed by the minimum requirements set by application goals for interrupt generation (parametric constraints) and network architecture capabilities.

The process of extracting an entity from sampled data is described in brief. An embedded node performs some signal processing operations after it samples real-time data using its sensors and extracts attributes from the output of these signal processing algorithms. These attributes are used to extract features that characterize a particular entity using either supervised or unsupervised classification techniques [89] [81] commonly used in data mining applications. The procedure used by classification techniques discretize attributes into intervals and each interval corresponds to a feature that is unique for each entity. For example, if the purpose of the application is vehicular detection using acoustic sensors, the data samples are input to an FFT algorithm that outputs the attribute *frequency* of the acoustic signals. Each vehicle (entity) can be characterized according to a range of frequencies and each range corresponds to a feature that distinguishes one vehicle from another. We assume that the sensors after processing sensed data identify entities through already existing data classification techniques and focus on the problem on detecting emergent clouds.

Parametric constraints to define a cloud are set by application goals that are defined using high-level specifications described in Chapter 2. These parametric constraints correspond to a minimum area covered by the cloud that has a density value higher than a threshold. For example, the entities are toxic gas clouds in the application example shown in Figure 1.1 and the schema for gas cloud detection in Figure 2.2 describes these parametric constraints where Lim_{gas} defines the density threshold, and the minimum area covered by the cloud is defined using the variables Lim_x and Lim_y, i.e. the minimum range of cloud in terms of the x and y coordinates of the network.

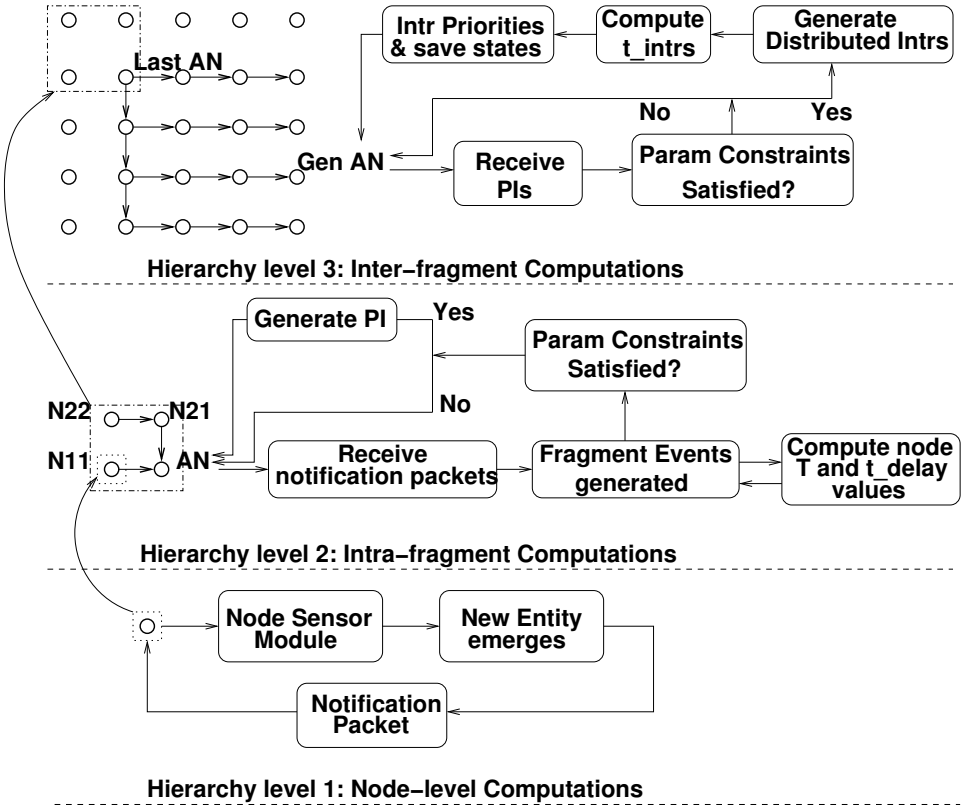

Figure 3.1: Algorithm for distributed Interrupt Detection

3.3.1 Node-Level hierarchy

The first level of hierarchy operates at the node level and the nodes detect an emergent entity. If the sensor notifies a node of an emergent entity, the node immediately processes the sensed data, computes the signature characteristics of the entity and forms a notification packet. A notification packet is generated due to data inconsistencies in the sampled signal. In this application scenario, data inconsistency is an outlier [89] that corresponds to the addition of a new entity to the Attribute Vectors (AVs) formed by the node after processing data. A notification packet is an outlier which may or may not be an emergent event. Individual nodes cannot make this decision at the bottom level of hierarchy due to insufficient data.

Referring to equation (3.1), the time taken by the sensor to detect an event and notify the sensor is given by t_{detect}. $t_{process}$ is the amount of time taken by the node to process the sample after sensor notification, update its AV and form the notification packet. The time taken for the node to detect the emergence of a new entity, process it and produce the notification packet is given by t_{prod}.

$$t_{prod} = t_{detect} + t_{process} \tag{3.1}$$

3.3.2 Intra-fragment level of hierarchy

The second level of hierarchy is called the intra-fragment level, where the algorithm performs intra-fragment communications and notification packets are collected at aggregation nodes (ANs). The ANs, that represent a default fragment, combine notification packets to form fragment events. After forming fragment events, the ANs determine if they are potential interrupts (PI) and filter out PIs from outliers based on certain rules that are defined by parametric constraints as shown in figure 3.1. A fragment event is not a PI if it does not satisfy the density constraint (e.g. Lim_{gas} in Figure 2.2). By collaborating with neighbors, emergent entities are distinguished from situations where an existing cloud of entity moves into the vicinity of a fragment and the fragment event is dropped. Noise and sensing errors are identified by defining a threshold for a percentage value. This percentage value is the ratio of the aggregated parametric value at the AN to a computed reference value. These parametric values depend on the sensor type and the reference value is a function of the number of

nodes in the fragment, sensor characteristics like range and sensing errors based on sensor abnormalities, and network granularity.

Since the nodes in the network are not synchronized with a global clock, time predictions have to be made at the ANs. Each AN captures the time instant when it receives a node event from an individual node and produces a time stamp τ_k using the node's timer module.

$$T_k = \tau_k - t_{delay\_k} \tag{3.2}$$

From these captured time stamps, the AN predicts the time instances of event detection at each individual node T_k by computing the latency t_{delay_k} given by equations (3.2) and (3.3). The term t_{comm_k} refers to communication time which is a function of baud rate and packet size, $l(path\_k)$ refers to the length of the path from the node k to the AN, t_{wait_k} refers to average waiting times at the input buffers of the forwarding nodes and t_{prod_k} refers to the time taken by the node to produce the node event which is given by equation (3.1). t_{prod_k} is computed at the first level of hierarchy and all other terms except t_{wait_k} are known.

$$t_{delay\_k} = t_{comm\_k} \times l(path\_k) + t_{wait\_k} + t_{prod\_k} \tag{3.3}$$

The t_{wait_k} values of all nodes in the fragment are computed using the time stamps of the individual nodes at the AN. The nodes are labeled with respect to path distance from AN. Hence t_{wait_k} is the waiting time of the node that is at a path distance k from AN within the fragment. Starting from the AN, the waiting times along the intra-fragment path of all the nodes are computed. For the AN, t_{delay_0} is equal to t_{prod_0} and all other terms including the waiting time is zero. For the next node which we term as AN-1, t_{wait_1} is zero since the length of the path is 1 and there are no forwarding nodes. For nodes that are x nodes away from the AN, a procedure is used to compute the wait times.

$$t_{wait\_x} = t_{wait\_x_{x-1}} + t_{wait\_x_{x-2}} + ... + t_{wait\_x_1} \tag{3.4}$$

The method to predict waiting time is given by equation (3.4) and (3.5). The term $t_{wait\_x_y}$ indicates the waiting time of node x at node y. The values of $t_{wait\_x_y}$ are sequentially computed starting with $y = x - 1$ and ending at $y = 1$. The r terms in

the equation are binary variables and hold either 1 or 0. An $r_a(b)$ value of 1 indicates that the waiting time of node a in node b was a finite positive term and the value of zero indicates that the waiting time was negligible since node b was not processing samples when it received data from node a in its input buffer and the waiting time of the data from node a at the input buffer of node b is negligible. The $r_a(b)$ values are computed using a simple algorithm that uses existing T_k and τ_k values.

$$t_{wait\_x_i} = \begin{cases} (\tau_x - \tau_i).r_x(i), & \text{if(i = x-1)} \\ [\tau_x + \sum_{j=i+1}^{x-1} wait\_x_j - \tau_i].r_x(i), & \text{otherwise} \end{cases} \tag{3.5}$$

The time taken for event nodes to reach the AN t_{intra_frag} is the difference between the time instance when the first node in the fragment generates an event and the time instance when the last node in the fragment reaches the AN. The time taken for an AN to generate a PI t_{prod_AN} depends on t_{intra_frag} and the AN processing time $t_{process\_AN}$. $t_{process\_AN}$ is the amount of time taken for an AN to produce a fragment event and determine if it is a PI.

$$t_{prod\_AN} = t_{intra\_frag} + t_{process\_AN} \tag{3.6}$$

3.3.3 Inter-fragment level of hierarchy

At the highest level of hierarchy, the ANs perform inter-fragment collaborations to decide if the PIs collectively can form an emergent event resulting in distributed interrupt generation. The AN that detects an interrupt, called AN_{gen}, broadcasts an interrupt generation packet and the relevant nodes start executing the ISR. This decision is made based on the area constraints set by application goals.

Since the purpose is interrupt detection with minimum communication, the ANs use the basic path structure for inter-fragment communication shown in Figure 3.1 to avoid flooding of PI packets. All ANs follow the same path procedure which ensures that PIs from an AN is not received multiple times at any other AN. The procedure is such that the PIs tend to diffuse to a particular direction.

Let AN_{last} be the last AN from which AN_{gen} received a PI before interrupt generation. After detecting an interrupt, AN_{gen} performs computations using the t_{prod_AN} values and the time stamp values to determine the AN with the highest latency called

AN_{lat}. AN_{lat} is the one of the first fragments that generates a PI and the nodes in this fragment are the first ones to detect emergent events. The interrupt latency t_{intr} is given by (3.7) and it depends on the time taken for AN_{lat} to generate a PI, the communication time between AN_{lat} & AN_{gen} and the time difference between the time stamp values of AN_{lat} and AN_{last}.

$$t_{intr} = \tau_{AN_{last}} - \tau_{AN_{lat}} + t_{prod\_AN_{lat}} + t_{comm\_AN}.l(path_{AN_{lat}}) \tag{3.7}$$

Hence, by combining equations (3.1), (3.3), (3.6), (3.7) and their dependencies, we compute the interrupt latency after the sensors detect the asynchronous inputs and this time delay depends on the sequence in which the sensor inputs are detected, entity dynamics, node processing times, baud rate, packet sizes and parametric constraints.

The accuracy of predicting the amount of time required to generate an interrupt t_{intr} is critical as it is helpful to compute the timing constraints during the process of building data models.

3.4 Experiments

The first set of experiments provides timing and power analysis of these middleware routines and network performance related measurements for a grid network of reconfigurable embedded PSoC processors. The second set of experiments analyze the performance of the event detection procedure that detects emergent entities.

3.4.1 Execution support routines

This section presents the measured execution time and power consumption of the implemented execution support. Measurements were done for a grid-type physical network of reconfigurable embedded PSoC processors [37]. PSoC is a system on chip, which offers an 8-bit microcontroller, flash memory for programs, SRAM memory for data, and programmable digital and analog cells, which are all integrated on the same silicon chip. PSoC's hardware reconfiguration capabilities are important for improving performance by customizing the architecture to the application needs. PSoC nodes were wired together in a grid and communicate with each other through UART modules. Tables 3.4.1 and 3.2 provides a detailed description of the power consumption and the execution times of all routines.

Definitions	# clock cycles	Power (mW)
Region	1,974	259.42
Target point	1,847 + 84 × (r - 1) + 112 × no_nodes	271.45
Path	453 + 84 × (r - 1) + 68 × n + 234 × p	251.65
Path rates	1,400 + 84 × (r - 1) + 86 × q	252.93
Actuation	1,626 + 84 × (r - 1)	251.96
Precision	1,378 + 84 × (r - 1)	240.06
Event_region	1,284 + 84 × (r - 1)	239.51
Event_param.	1,843 + 84 × (r - 1)	251.715

Table 3.1: Timing and power consumption analysis of execution support for command packets

As there are cases in which the execution time of the methods is variable, for example definition of a target point, the execution time was split to reflect the constant component and the variable parts. For example, the number of clock cycles of target point definition is $1,847 + 84 \times (r-1) + 112 \times no\_nodes$. The constant overhead takes 1847 clock cycles. The other two parts represent timing of loops, where it takes 84 and 112 clock cycles respectively to execute each run of the loop, and the loops are run $(r-1)$ and *no_nodes* times respectively. The referred region is the r-th entry in a node's data structure. *no_nodes* denotes the number of nodes in that specific region. The average power consumption was measured by executing continuously each routine on PSoC. A multimeter connected in series with the supply pin of PSoC and the power source indicated the current drawn by PSoC. The average consumed power resulted by multiplying the measured current with PSoC operating voltage.

Table 3.4.1 indicates the execution time (number of clock cycles) and power consumption of the command packets for defining regions, target points, paths, path rates, actuation, sensing precision, and events. According to the scheme used by the nodes to broadcast data, the nodes transmit the packets to the neighboring node(s) after updating the local data structure. Parameter n denotes that the node is the n-th node in the path. Value p is the size of the packet. Parameter q denotes that the path associated is the q-th path in the data structure of the region.

Function	# clock cycles	Power (mW)
Sense	64,720	571.86
Process	277 +Comm._exec.+ Forward._function	230.9
Check_event	1,038 + 84 × (r - 1) + 84 × (t - 1)	242.83
Actuation	268 + 84 × (r - 1)	242.21
Transfer_from_buffer	225 + Process	253.46
Select_path	1,233 + 75 × (r - 1) + 91 × (q - 1)	243.38
Forward_data_packet	1,087 + 75 × (r - 1) + 35 × (p - 1)	227.21
Forward_event_packet	848 + 84 × (r - 1)	236.72

Table 3.2: Timing and power consumption analysis of execution support for data packets

Table 3.2 illustrates the clock cycles analysis and power consumption measurements for the temperature sensing routine. In *Check_event* and *Actuation*, r defines that the current region is the r-th entry in the data structure for events and actuation, respectively, and t denotes that the current region is the t-th entry in the data structure of the target points for all regions. In *Select_path*, the parameter r indicates that the current region is the r-th entry in the data structure, and variable$q$ denotes the number of paths eliminated while selecting the path. For routine *Forward_data_packet*, parameter r indicates that the region name in the data packet is the r-th entry in the data structure that defines the path, and variable p denotes that the path name in the data packet is the p-th path defined for that region in the data structure. For *Forward_event_packet*, r defines that the region in the event packet is the r-th entry in the data structure that stores the target points for all regions defined for the node.

The timing overhead of the routines is low. It increases linearly with the number of monitored DARs, number of paths, the packet size, and the position of a node on the paths. The power consumption is around 250mW for all routines with the exception of the sensing module, for which it is higher.

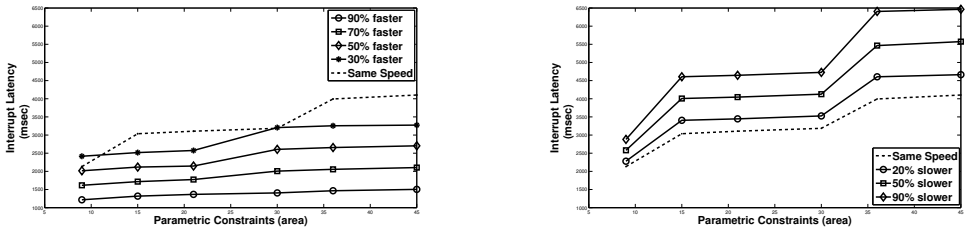

Figure 3.2: Interrupt detection time vs. parametric constraints

3.4.2 Performance of the event detection algorithm

Experiments were performed by producing emergent clouds moving and changing their shapes and sizes over time on a grid type PSoC [37] network simulator containing 100 nodes with a granularity of $3m$. The sensor used was a microphone array that localizes entities producing sound. Sensor abnormalities were captured through real-time experiments and incorporated in the simulator. Entities were identified using a pre-defined, supervised discretization technique that used frequency intervals.

Emergent data clouds were formed starting with a couple of nodes and expanding over the network with different dynamics. The proposed algorithm predicted interrupt latency t_{intr} and the percentage prediction error was computed. It was observed that interrupt latency and prediction error are independent of cloud size and position for a given set of parametric constraints and cloud dynamics. This behavior is observed due to the uniform path structure used by the algorithm. Interrupt is detected dynamically by the first node that satisfies the parametric constraints. Hence, we show experimental results on the same emergent cloud by varying dynamics and parametric constraints.

Experiments were performed on 8 different clouds of entities with different sizes, positions and dynamics. Distributed interrupts were successfully generated for all clouds.

The cloud dynamics are measured with respect to the node processing capabilities. The processing capabilities correspond to the time taken to process sampled data. Tables 3.3 and 3.4 provide experimental results for different cloud dynamics for the same cloud. Column 1 in both tables shows cloud dynamics relative to node processing speed. Columns 2 to 7 show interrupt generation time in table 3.3 and percentage prediction error in table 3.4 for different parametric constraints. The parametric

Cloud Dynamics wrt node processing power	Interrupt Generation Time (msec)					
	$9m^2$	$15m^2$	$21m^2$	$30m^2$	$36m^2$	$45m^2$
1	2	3	4	5	6	7
90% faster	1216	1317	1366	1406	1466	1503
70% faster	1616	1717	1775	2006	2056	2103
50% faster	2016	2117	2146	2606	2656	2703
30% faster	2416	2517	2575	3206	3256	3273
Same Speed	2130	3036	3106	3186	3993	4103
20% slower	2280	3406	3446	3526	4605	4663
50% slower	2580	4006	4046	4126	5465	5573
90% slower	2880	4606	4646	4726	6405	6463

Table 3.3: Prediction of time taken to generate interrupts

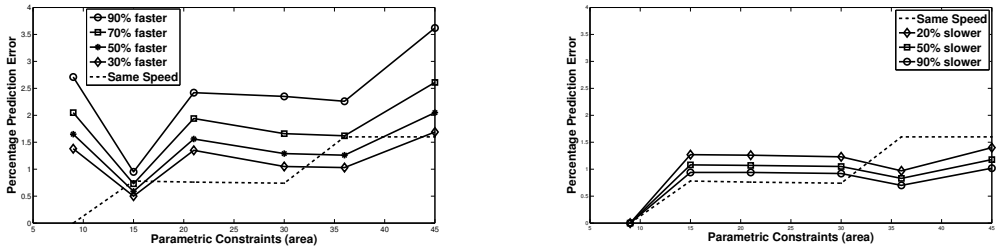

Figure 3.3: Percentage prediction error vs. parametric constraints

constraints correspond to minimum area that need to be covered by the entity to generate an interrupt. We analyze the change in interrupt latency and percentage prediction error with cloud dynamics and parametric constraints with the help of the plots in figures 3.2 and 3.3. Percentage prediction error corresponds to the difference between the t_{intr} value computed by the algorithm to the actual interrupt latency time under the assumption that the nodes are clock synchronized.

Figure 3.2 is used to analyze changes in interrupt latency with changes parametric constraints for different dynamics. Figure 3.2a includes clouds with dynamics faster than node processing speed and figure 3.2b includes clouds with dynamics slower than node processing speed. It is observed that the slope in figure 3.2 remains constant for a specific range of areas. The slope depends on the value of $l(path_{AN_{lat}})$ in equation (3.7). We observe an abrupt change in the slope for an increase in $l(path_{AN_{lat}})$. For example, the values of $l(path_{AN_{lat}})$ increases with areas ranging from $20m^2$ to $30m^2$ in figure 3.2a and also areas ranging from $10m^2$ to $15m^2$ and $30m^2$ to $35m^2$ in figure 3.2b. It is observed that if the cloud dynamics is faster than node processing power, the change in slope is less apparent comparatively and is almost non-existent for the case that is 90% faster as shown in figure 3.2a. The change in slope is more pronounced for slower cloud dynamics as shown in figure 3.2b.

Figure 3.3 is used to analyze changes in prediction errors for changes in parametric constraints with different cloud dynamics. Figure 3.3a includes clouds with dynamics faster than node processing speed and figure 3.3b includes clouds with dynamics slower than node processing speed.

Both figures 3.3a and 3.3b show that the algorithm works consistently for different cloud dynamics with a few exceptions. It was observed that the errors due to radio retransmissions is more dominant. The percentage error for parametric constraints

Cloud Dynamics wrt node processing power	Prediction Error (%)					
	$9m^2$	$15m^2$	$21m^2$	$30m^2$	$36m^2$	$45m^2$
1	2	3	4	5	6	7
90% faster	2.71	0.95	2.42	2.35	2.26	3.62
70% faster	2.05	0.73	1.94	1.66	1.62	2.61
50% faster	1.65	0.59	1.56	1.29	1.26	2.05
30% faster	1.38	0.50	1.35	1.05	1.03	1.69
Same Speed	0.00	0.78	0.76	0.74	1.66	1.60
20% slower	0.00	1.27	1.26	1.23	0.97	1.40
50% slower	0.00	1.08	1.07	1.05	0.83	1.178
90% slower	0.00	0.94	0.94	0.92	0.70	1.018

Table 3.4: Percentage prediction error

less than $10m^2$ is high for faster clouds due to the above reason. For the same parametric constraints for slower clouds, the slow clouds produce a prediction error of zero. For these cases, AN_{gen} is the same as AN_{lat} and the algorithm produces predictions based on measurements local to this node and error due to retransmissions is zero. For higher values of min area, other errors like predicting waiting times t_{wait_k} also make contributions. As expected, the algorithm produces higher percentage errors for cloud dynamics faster than processing speed due to longer waiting times at node input buffers. The percentage error is lowest for clouds which have dynamics equal to node processing time.

3.5 Conclusions

This chapter describes the execution platform that defines middleware routines to run network-level applications on the grid-type sensor network. This execution support is used to run all experiments to analyze all algorithms discussed in this thesis. The middleware routines can generate command packets that specify network parameters like regions, data paths, target points, sampling rates, sampling precision, event threshold values and actuation procedures. Timing and power analysis of these middleware routines and network performance related measurements are discussed in the experiments section, and are used to build an efficient SystemC simulator that closely emulates the physical network.

The execution support also has the capability to detect emergent entities in time and space in the form of cloud of physical entities. The novel concept of distributed interrupts is used for detecting these emergent events. These emergent entities are detected using asynchronous sensor inputs sensed by embedded nodes which are not synchronized by a common clock. The three-level hierarchical algorithm was able to correctly generate a distributed interrupt or an emergent event for different cloud dynamics, area and size under a wide range of parametric constraints. Interrupt latencies are accurately predicted and percentage prediction error ranges from 0 to 2.7%.

Chapter 4

Online Adaptation Policy Design of Parameterized Execution Platform to Track Quasi-Static Entities[1]

4.1 Introduction

This chapter discusses an optimization model that automatically devises the adaptation policies of the reconfigurable sensor nodes to address the goals and constraints of the goal-oriented descriptions. The procedure discussed in this chapter targets applications where entities are typically quasi-static in nature, e.g. environmental monitoring applications where temperature throughout the domain has to be constantly monitored and data model parameters are constantly updated. These applications define the global tasks and goals that must be achieved by a networked distributed system through combined operation of its embedded nodes. As a result, two main design challenges emerge, (i) each sensor node has to efficiently sense, process, and network under a wide range of performance requirements, while (ii) only scarce hardware, bandwidth, and energy resources are available (to keep the cost low). The performance requirements are specified by the user using the schema-based, high-level specifications described in Chapter 2. For example, the performance requirements for the application example described in Figure 1.1 was specified by the schema in Figure 2.1. In this specification, the area domain (variables DOM_x and DOM_y) is

[1]This work was published in [1, 10]

specified and pollution within that domain has to be minimized, and timing constraints were set where the delay with which data has to be available for building data models should not exceed the specification provided by variable Lim_{req}. The second design challenge directly relates to the application-specific constraints. These constraints are imposed by the schemas that correspond to entity monitoring and data sampling. For example, the specifications for gas cloud monitoring in Figure 2.2 and for temperature sensing in Figure 2.3 describe predicates to describe a few threshold requirements that the acquired data needs to meet for successfully building data models to represent the phenomena. The embedded network infrastructure needs to optimally utilize the available resources to sample sufficient amount of data and make this data available for building data models. The reason for targeting applications with quasi-static entities was to study how the optimization model switches between network resource parameters and adjusts the cost function depending on varying goals and resource constraints without worrying about the different facets that describe the dynamic nature of the entities. Adaptation strategies for dynamically changing entities are described in Chapter 5. Present design methods are insufficient for tackling the two challenges. Methods have only limited capability for co-optimizing the sensing, processing, and communication subsystems of embedded nodes. Also, few methods exploit the flexibility of networked reconfigurable architectures to produce low-cost yet efficient designs for a broad range of requirements.

The procedure discussed in this chapter devises online reconfiguration policies that control the characteristics of the sensor nodes and data routing to tackle the two challenges. The algorithm uses two steps: (i) Design Point generation, and (ii) adaptation policy design at node and network level hierarchies.

The first step produces Design Points (DPs) that are used by the adaptation model to switch between network parameters to improve performance of the network infrastructure. These design points are generated for each network parameter module depending on the application goals and resource constraints imposed by the application. Each DP specifies a unique performance-cost trade off. For example, each DP would correspond to different communication bandwidth values that can be selected by the optimizer. The goals and application constraints describes by the schemas in Chapter 2 would provide the range of bandwidth values that can be used and the procedure can generate n number of intermediate values that can be used for data communication. The performance-cost trade offs correspond to different

communication speed-power consumption ratios and other performance factors. For example, higher bandwidth values would improve latency, but the embedded nodes would consume high power at the communication modules, corresponding to latency-power trade offs. Transmitting large amounts of data with high bandwidths can flood the network, resulting in data loss, which corresponds to latency-data loss trade offs. The DP generation procedure uses a systematic simulated-annealing based approach using Multimode Dataflow Graphs (MDGs) to capture the multiple operation modes of the sensor nodes. These MDGs are graphical representations that are built using the performance resource constraints specified by high-level specifications specified in Chapter 2.

Using a Linear Programming solver describing adaptation as a Continuous-Time Markov Chain, the second step calculates the switching rates between alternative DPs and possible communication paths to optimize performance requirements of the application. The executable code derived from high-level descriptions, along with the cost function derived from the application goals and the DPs are available at individual nodes. After adjusting the coefficients in the cost function based on the application goals specified by the corresponding schema and computing the transition probabilities to switch between available DPs, the node runs this executable code on the execution support and sufficient amount of data is provided to build accurate data models.

4.2 Related Work

The sensor network research community has traditionally studied network routing protocols [69] without focusing much on design methods for networks of reconfigurable sensor nodes. Fortunately, there is some similarity between sensor networks and Networks-on-Chip (NoCs) [70, 71, 72]. For both, the performance of the networking infrastructure is critical for the overall system performance. Algorithms for NoC path selection, mapping of cores, and TDMA time slot allocation are discussed in [73]. Methods for communication topology mapping, and topology and protocol selection are offered in [74, 71, 68, 35]. Ascia *et al.* [74] present a Genetic Algorithm for finding the communication topology, protocols, and priorities of mesh-based NoCs with static routing of communication packets, and the worst-case performance is estimated using a trace-based simulator. Bertozzi *et al.* [71] suggest a communication network

design flow that includes topology mapping, selection, and generation for minimizing objectives, like area, power, hop delay, and bandwidth of various topologies, like mesh, torus, hypercube, etc. The designer specifies the mapping of the functionality to cores and the routing function, e.g., dimension ordered, minimum-path, traffic splitting across minimum-paths, and traffic splitting across all paths. The methods perform a greedy mapping of the communications between cores followed by solving multi-commodity flow equations for finding the splitting ratios across multiple paths. Designs are simulated with cycle and signal level accuracy. Murali *et al.* [68] propose a methodology to map different use-cases to reconfigurable NoCs, so that the constraints of each use-case is met. Multiple use-cases can run in parallel. The core mapping remains the same when switching from a use-case to another but the communication paths and TDMA slots are reconfigured to match the new use-case. Guz *et al.* [75] discuss allocation of non-uniform link capacities under QoS timing constraints. Wormhole routing is considered due to its simplicity, small latency, and reduced buffer space requirements. Performance is estimated using analytic models. Capacity allocation assigns to links small chunks of link capacity guided by bandwidth sensitivity. The related work on NoCs offers very interesting methods for designing the networking infrastructure, but does not tackle the interdependency between the design of reconfigurable nodes and the network. Exploiting the adaptation opportunities of networked embedded nodes reduces data loss, latency, and power consumption without increasing cost.

4.3 Optimization Model

This section discusses the optimization model that automatically devises the adaptation policies of the reconfigurable sensor nodes to address the goals and constraints of the goal-oriented descriptions. The challenges addressed by this optimization model involve reliable sensing, processing, and networking to meet application goals by optimally utilizing the limited resources that are available to run the application.

The optimization procedure uses two steps: (i) Design Point generation, and (ii) Adaptation Policy Design at the node and network level hierarchies.

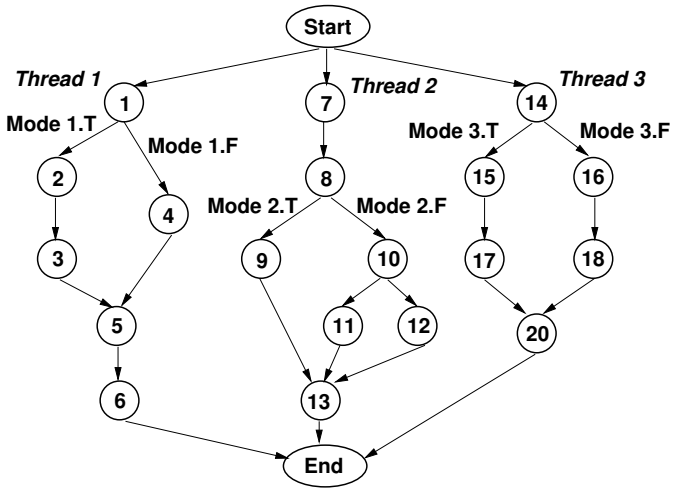

Figure 4.1: Multimode Dataflow Graph

4.3.1 Design Point Generation

Chapter 2 discusses the schema-based high-level descriptions that enable the use to specify application goals, related procedures and resource constraints. These descriptions are used to build Multimode Dataflow Graphs (MDGs). MDGs are polar graphs that consist of multiple parallel threads spanning between the start and end nodes, as shown in Figure 4.1. Each thread is composed of nodes connected through arcs. A node describes an algorithmic component, e.g., function, subroutine, etc. Arcs express the required sequencing and data dependencies between nodes. Nodes communicate through shared memory. Nodes can share hardware resources for their implementation. Each thread might include multiple modes. A mode represents a pair of polar sub-graphs, which are executed in mutual exclusiveness depending on the value of the condition associated to the mode. Condition values change continuously. For example, in Figure 4.1, Nodes 2 and 3, and Node 4 define the two complementary branches of Mode 1. The first branch is selected if the condition is true (labeled $Mode1.T$), and the second branch if the condition is false (labeled $Mode1.F$). Modes can be nested.

The method to generate design points is based on the algorithm proposed in [35], which handles systems with multiple operation modes. The algorithm co-optimizes all modes together as opposed to traditional methods, which consider only one mode at a time, usually starting with the most performance-constrained mode [68]. As shown in [68], traditional strategies can result in hardware over-design as high as

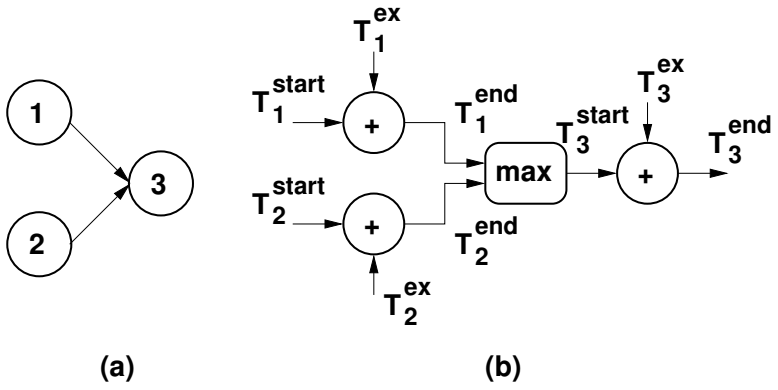

Figure 4.2: Scheduling using MDGs

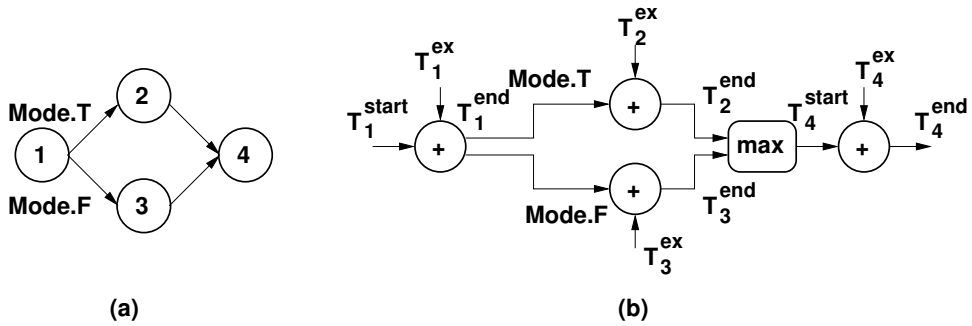

Figure 4.3: Scheduling with multiple modes

90% compared to the optimal solutions.

The algorithm for generating multiple DPs is based on Simulated Annealing (SA). It finds the resource mapping and scheduling of MDG nodes for different hardware resource sets. The cost function is a weighted sum of the performance attributes. The exploration loop performs two types of moves: it changes with probability p the mapping of a node to resources, and it modifies with probability $(1-p)$ the order of execution of two nodes mapped to the same hardware resource. Hence, the algorithm conducts simultaneous mapping and scheduling, which offers better solutions than having two separate steps [35].

Using a given MDG, the procedure performs simultaneous mapping and scheduling by evaluating the performance attributes of an MDG's implementation, e.g. latency, power consumption, etc. The variables in the MDGs denote the performance values of the blocks in the implementation. In Figure 4.2(b), variable T_1^{ex} is the execution time of Node 1 for its current hardware binding (for example, the execution time of a task for certain resources). Similarly, variable T_2^{ex} is the execution time of Node 2,

and so on. Variables T_i^{start} and T_i^{end} are the start and end times of Node i. Operator nodes correspond to operators and mathematic functions, e.g., plus, maximum, etc., and are used to compute the overall performance attributes of an implementation based on the node attributes and the specific scheduling order. For example, Figures 4.2, 4.3, and 4.4 are used to evaluate the overall latency of an implementation. The DPs that are eventually generated should evaluate to a latency using the above procedure while keeping application specific goals in mind. For example, a range of DPs should be selected such that the evaluated latencies should satisfy the timing constraint Lim_{req} in the schema in Figure 2.1 while meeting the constraints related to energy consumption of the specification. Similar graphs can be set-up for attributes other than latency and power consumption. An overall performance attribute is computed by pre-order traversal of the graph. Hence, performance estimation using this procedure has module-level accuracy. Alternative DPs, each corresponding to the T_i^{ex} values and equivalent values for other performance attributes are generated by the algorithm.

The procedure also offers the benefit of simplifying the combined expression of mapping and scheduling. Figure 4.2(b) illustrates the structure built to express the data dependencies between nodes 1, 2, and 3 in Figure 4.2(a). The graph shows that the start time T_3^{start} must be larger than both end times T_1^{end} and T_2^{end} of nodes 1 and 2. Moreover, for each node i, its end time T_i^{end} is the sum of its start time T_i^{start} and execution time T_i^{ex}. Figure 4.3(b) illustrates the sub-graph set-up for the two modes in Figure 4.3(a). The semantics are similar to the previous case with the difference that the branches for the unselected modes propagate the value $-\infty$ to the *max* operator node. Thus, the unselected modes do not affect the overall performance. If the hardware binding of a node changes then the value of the T^{ex} variable changes too. The scheduling order between two nodes mapped to the same resource is expressed through a dotted arc, as shown in Figure 4.4(b). If Node 2 in Figure 4.4(a) is executed before Node 1 then the dotted arc in the PM reflects the ordering constraint. The SA loop changes the execution order between the two nodes by modifying the direction of the dotted arc. A dotted arc is dynamically added whenever two nodes share the same resource, and is removed if the mapping for one of the nodes changes.

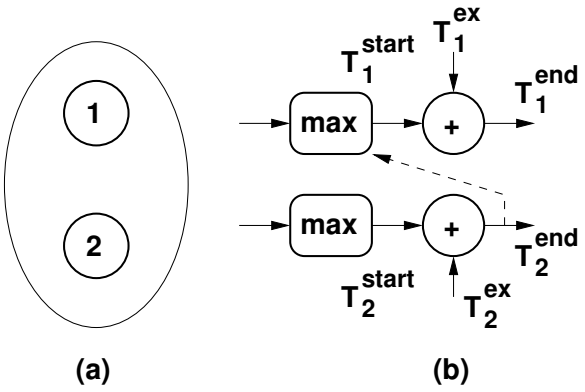

Figure 4.4: Scheduling for resource sharing

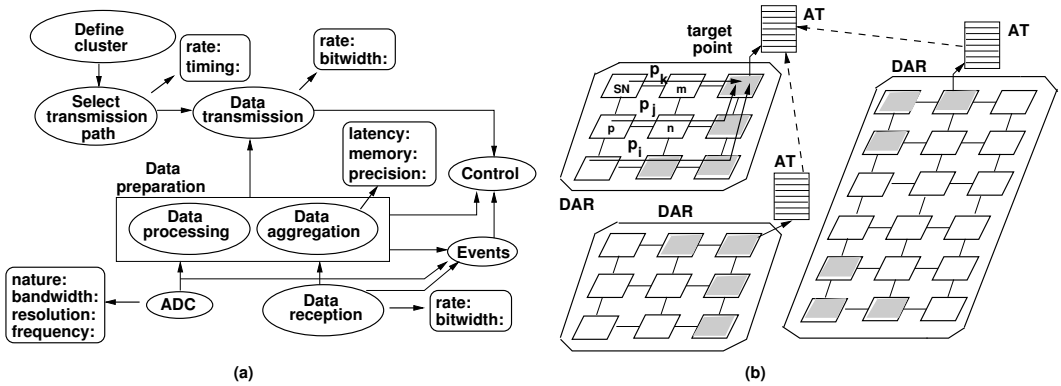

Figure 4.5: Programmable sensor node and grid network of reconfigurable nodes

4.3.2 Adaptation Policies by the Data Communication Network

The Data Communication Network (DCN) is a grid network of parameterized nodes called Sensing Nodes (SNs), as shown in Figure 4.5(b). As explained later in this subsection, grid networks simplify the estimation of the transmission delays and the prediction of the data loss of a node. Handling networks of different topologies requires expending the approach, so that the grid network acts as a virtual layer between the physical network and the optimization model. Then, an additional step can map the links of the virtual network to physical routing links similar to the method discussed in [66].

This step calculates the switching rates between alternative DPs and possible communication paths to optimize the cost function that minimizes data loss and latency, timing constraints, scalability, precision and resource constraints. The coefficients

of the cost function are adjusted depending on application goals. Application goals are specified during run time and the rate of utilizing DPs over time depends only on currently available resources and application requirements, and is independent of history. Hence, equations are formulated in the form of Continuous-Time Markov Chains (CTMC) to solve the optimization problem and the transition probabilities computed to switch between DPs depend only on the current state and is independent of the previous state of the process. The optimization equations are formulated for SN-level production and consumption of data, and network-level data between SNs.

A. SN-level production and consumption of data

At the SN-level, the operation of the node routines are modeled as producers that place data into a common buffer, and consumers that remove data from the buffer. The buffers represent the local memory of an embedded sensor node. Each thread handling sensing data and data packets (i.e. threads shown in Figure 2.4(a)) has one producer and one consumer. Producers are of two kinds: *Producers-S* represent data sensing through physical sensors, and *Producers-N* correspond to data packets from the network. Consumers model the data packet output to the network as well as any local data processing, e.g., filtering and local data aggregation. Consumers remove data from the buffer without adding data back to it. Figure 4.6(b) illustrates the model.

Example: The thread for temperature sensing (Figure 2.5(a)) has one related Producer-S and one Consumer in the DCN. This producer corresponds to the code that starts with reading the output of the temperature sensor and ends with placing the read data into the local memory. The consumer represents the code that transmits one data packet. Similarly, one Producer-S and one consumer are for the routines in Figures 2.5(b)-(c). The routine in Figure 2.5(d) has one Producer-N corresponding to the code that reads one packet from the network and one consumer. The producer describes receiving bits from the network, assembling data packets, identifying the values in packets, and placing the values into buffers.

Each producer (consumer) can have different data production (consumption) characteristics, e.g., speed, power consumption, production rate and data precision, depending on the specific implementation on the reconfigurable architecture. An implementation is called Design Point (DP). Hence, each producer (consumer) is described by a set of distinct DPs. Selection of the DP set is performed using the methods

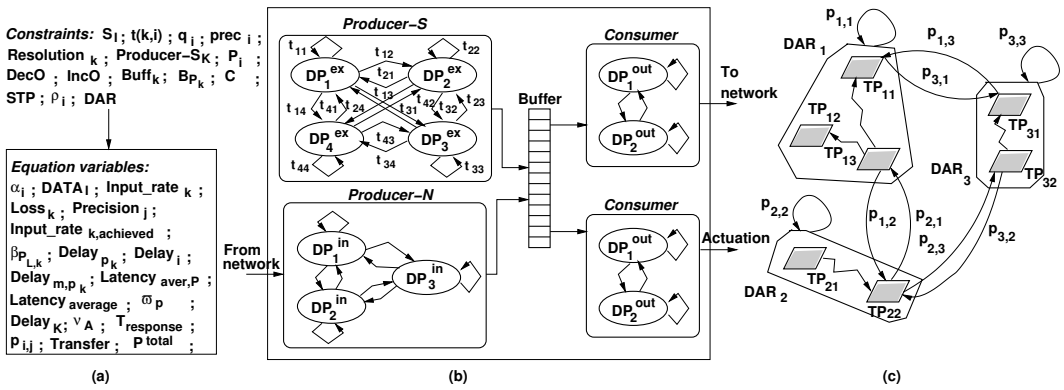

Figure 4.6: Model constants and variables, nodes and clusters of Data Communication Network

described in [49, 57]. During operation, producers and consumers switch among DPs to adapt to the specific conditions and requirements. The modeling of the local, SN-level data production and consumption must capture the switching between different DPs.

The SN-level data production and consumption is modeled as Continuous-Time Markov Chains (CTMC):

$$Producer_I(Consumer_I) = (S_I, A, A(i), t, K, r) \tag{4.1}$$

For producer (consumer) I, S_I is the set of states, A is the action set, and $A(i)$ are the actions associated to state $i \in S_I$. $t(i,j,a)$ is the transition rate if action a is chosen for transitioning from state i to state j. K is the number of reward criteria. $r_k(i,a)$ is the reward rate if action a is selected for state i. α_i is the steady-state probability of state i.

Example: In Figure 4.6(b), *Producer-S* has four states, each state being a different design point DP_i. Similarly, *Producer-N* has three states. The reward criteria are the performance attributes of the DPs, such as latency, power consumption, and input data rate. The action set A has only one action, which is defined by the fixed functionality of a DP. Hence, for brevity, actions are not indicated in the following CTMC equations. The action set A would have multiple actions, if each SN has multiple alternative routines to execute the node functionalities, like alternative sampling and processing methods. Then, the actions would represent the alternative routines that the node can select. The overhead, e.g., time and power, for switching states is

not expressed in the model as it is low for a reconfigurable architecture.

The execution rates of DPs are expressed for two situations, if the producers (consumers) do not share or share resources. In the first case, the following equations describe each state i of producer (consumer) I:

$$t(i,i)\ \alpha_i - \sum_{\forall k \in S_I} t(k,i)\ \alpha_k = 0 \tag{4.2}$$

$$\sum_{\forall i \in S_I} \alpha_i = 1, \qquad \alpha_i \geq 0 \tag{4.3}$$

Equation (4.2) correlates the rate of using DP (state) i to the rate of using DP k and the transition probability from k to i. Equation (4.3) indicates that one DP is always used for a thread.

If the producer (consumer) threads are executed on shared resources, e.g., shared processor, then the equations (4.3) for all threads must be eliminated, and replaced with the following (single) equation:

$$\sum_{\forall S_I} \sum_{\forall i \in S_I} \alpha_i = 1 \tag{4.4}$$

The equations indicate that the DPs of all threads are executed in mutual exclusion on the shared resource.

Assuming that each state i of a producer (consumer) I produces (consumes) data at a rate q_i ($q_i \geq 0$) then the average amount of data produced (consumed) in a unit of time is:

$$DATA_I = \sum_{\forall i \in S_I} \alpha_i q_i \tag{4.5}$$

The average input data rate (sensing precision) of SN K is:

$$Input\_rate_K = \sum_{j \in Producer - S_K} DATA_j \geq Resolution_k \tag{4.6}$$

where set $Producer - S_K$ is the producer set for all sensing routines of SN K.

Example: The second equation in Figure 2.1 states that the precision in time of data sensing (e.g., the time interval between consecutive sensor readings) has to be less than Lim_{req}, otherwise the corresponding time period is added to sequence

Delay. Hence, the precision in time for gas and temperature sensing has to meet the constraint. Thus, the input rate of the two sensors must meet the requirement $Input\_rate \geq \frac{1}{Lim_{req}}$.

The amount of data generated by all producers of an SN has to be larger than the amount of data consumed by all its consumers:

$$\sum_{\forall I \in Producers} \sum_{\forall k \in S_I} DATA_k \geq \sum_{\forall J \in Consumers} \sum_{j \in S_J} DATA_j \tag{4.7}$$

Assuming that the precision is cumulative, the bitwidth precision of a producer I is:

$$Precision_I = \sum_{\forall i \in S_I} \alpha_i prec_i \tag{4.8}$$

Example: According to the third equation in Figure 2.3, the bitwidth precision $Precision$ of the temperature sensors has to be larger than Lim_{prec} to avoid the generation of an event and a response $Action_1$.

The data loss per unit of time at node K is the difference between its overall production and consumption rates:

$$Loss_K = \sum_{\forall i \in S_{K,producers}} \alpha_i \; q_i - \sum_{\forall i \in S_{K,consumers}} \alpha_j \; q_j \tag{4.9}$$

The buffer capacity of any node K must be larger than the worst-case buffer space requirement:

$$\sum_{\forall i \in DecO(S_{K,prod.})} \alpha_i \; q_i - \sum_{\forall j \in IncO(K, S_{consumers})} \alpha_j \; q_j \leq Buff_K \tag{4.10}$$

Sets $DecO$ and $IncO$ are the set of states of the producers and the set of states of the consumers, respectively, for which data keeps accumulating in the internal buffer. In the worst case, input data is coming at the highest rates of the producers while data is consumed at the lowest rates of the consumers. The states in sets $DecO$ and $IncO$ include the DPs with the top k largest input rates and the DPs with the l lowest output rates, such that $\sum_{i \in DecO} \alpha_i \; q_i \geq \sum_{j \in IncO} \alpha_j \; q_j$. Set $DecO$ is ordered in the decreasing order of the rates, and set $IncO$ in the increasing order of the rates. Values k and l are the values for which the difference in equation (4.10) is maximum.

Data loss reduces the resolution that is achieved at the target point, where the sampled data is collected for decision making. Assuming that the loss at a SN is equally distributed among its producers, the achieved input rate of a SN is as follows:

$$Input\_rate_{K,achieved} = (\sum_{j \in Producer - S_K} DATA_j) - \frac{Loss_k}{Cardinality(Producers)} \tag{4.11}$$

and

$$Input\_rate_{K,achieved} \geq Resolution_k \tag{4.12}$$

where $Cardinality(Producers)$ is the total number of producers, which includes Producers-S and Producers-N. Note that equation (4.6) is required for accurate node-level decisions, and equation (4.11) is needed for precise global decisions.

B. Network-level data flow

The network-level data flow is modeled as a Data Aggregation Pattern (DAP). A DAP is a set of paths so that every SN is connected to a target point in the network. An SN can use multiple paths. Figure 4.5(b) illustrates a DAP including paths p_i, p_k, and p_j. SN n receives data from SN m through path p_i, and from SN p through path p_k. DAPs help expressing the communication parameters of connected SNs, e.g., communication load and delay.

The optimization model satisfies the following data conservation rule: the data output by one SN is entirely received by the inputs of the SNs connected to it. Lossy connections can be modeled by transfer functions of links. Without affecting the generality of the method, the paper considers only lossless links. As in Figure 4.5(b), for link L between any SNs m and n, set P_L is the set of all paths $p_{L,k}$ that use link L. B_{p_k} is the bandwidth and $\beta_{p_{L,k}}$ is the rate of using path p_k. Using equation (4.5) and the data conservation rule, the following equation states that P_L should have sufficient bandwidth for the communication between SNs m and n:

$$\sum_{\forall i \in S_{consumer_m}} \alpha_i q_i = \sum_{\forall j \in S_{Prod-N_n}} \alpha_j q_j \leq \sum_{\forall p_{L,k} \in P_L} \beta_{p_{L,k}} B_{p_k} \tag{4.13}$$

and,

$$\sum_{\forall p_{L,k} \in P_L} \beta_{p_{L,k}} = 1, \qquad \beta_{p_{L,k}} \geq 0 \tag{4.14}$$

The delay of path p_k is the average latency of all its SNs:

$$Delay_{p_k} = \sum_{\forall m \in p_k, I \in m} \sum_{i \in S_I} \alpha_i \; Delay_i \tag{4.15}$$

The delay of a node m on path p_k is:

$$Delay_{m,p_k} = \sum_{\forall n \in Succ(m,p_k), I \in n} \sum_{i \in S_I} \alpha_i \; Delay_i \tag{4.16}$$

where SN n follow node m in the path p_k towards the target point.
The latency of DAP P is equal to the maximum delay of its paths $p_k \in P$:

$$Latency_{aver,P} = \max_{p_k \in P} Delay_{p_k} \tag{4.17}$$

If a target point has a collection C of alternative DAPs then the average latency is as follows:

$$Latency_{average} = \sum_{\forall P \in C} \omega_P Latency_{aver,P} \tag{4.18}$$

$$\sum_{\forall P \in C} \omega_P = 1, \qquad \omega_P \geq 0 \tag{4.19}$$

ω_P is the utilization rate of DAP P.
The delay of receiving data from node K at the target point is:

$$Delay_K = \sum_{\forall P \in C, \forall p_i \in P s.t. K \in p_i} \omega_i \; Delay_{K,p_i} \tag{4.20}$$

where P is an alternative DAP in collection C, and p_i is the path in DAP P, so that SN K is part of path p_i.

Example: The second equation in Figure 2.1 requires that the pollution data acquired for every path segment should be available at the target point in less than Lim_{req} time, otherwise the data is considered obsolete. Hence, delay $Delay_K < Lim_{resp}$ for every SN K.

The rate at which the sensing data acquired by SN K is received at the target point TP is the minimum of the rate expressed in equation (4.11) and the experienced communication delay for the node:

$$Input\_rate_{K,TP} = \min(Input\_rate_{K,achieved}, \frac{1}{Delay_K}) \tag{4.21}$$

The following equations formulate the expressions used in expressing the goal of an application. The average power consumption of the network is the total average power consumption of all SN K in the network:

$$P^{total} = \sum_{\forall K} \sum_{\forall S_I \in K} \sum_{i \in S_I} \alpha_i \ P_i \tag{4.22}$$

The cost function minimizes a weighted sum describing the overall data loss, power consumption and delay:

$$Cost = \min(\gamma Loss + \delta \ P^{total} + \mu \ Latency_{average}) \tag{4.23}$$

In summary, the step for optimizing the DCN parameters uses the equations (4.2)-(4.23) to optimize the dynamic behavior of a SN network. Figure 4.6(a) enumerates the constants and variables of the model. The procedure calculates the steady-state probabilities and transition rates of the design points (DPs) and communication paths to optimize a cost function that expresses the application goals, data loss, latency, and power consumption. The step also considers the constraints of the architecture, such as available memory.

The discussed model differs from another Markov Decision Process-based optimization discussed in [65]. The approach in [65] focuses on maximizing the lifetime of sensor nodes by tuning the processor voltage and frequency and sensing frequency. In contrast, at the sensor level, the proposed model also describes memory size constraints, which is important as many embedded architectures have small data mem-

Ex.	Best-case			Worst-case			Diff. (%)	Exec. time (sec)
	List sch.	Opt.	Imp. (%)	List sch.	Opt.	Imp. (%)		
SN 16	82	41	50	136	108	20	62	33
SN 24	154	84	45	210	180	14	53	34
SN 32	173	156	9.8	242	215	11	27	112
SN 48	522	296	43	510	353	30	16	97
SN 64	633	440	30	852	510	40	19	225.30

Table 4.1: Design Point generation

ories. In addition, the model formulates data communication loss and delays, which are main aspects for acquiring accurate data for decision making.

4.4 Experiments

The experimental part presents results for (i) Design Point (DP) Generation, (ii) DCN network parameter optimization, and (iii) a heat source tracking application implemented on a wired grid network with up to 25 PSoC embedded processors [37]. The PSoC chips communicate with each other through UART modules. The goal was to study the speed - memory - power consumption trade-offs explored starting from goal-oriented specification, and to characterize the time and power consumption overhead of the execution support. The experimental results for execution support middleware routines can be found in appendix A.

4.4.1 Design Point Generation

The first set of experiments studied the quality of DP generation using the proposed method. MDGs with 16, 24, 32, 48, and 64 nodes have been considered. Each MDG had three modes. Five different resource sets were considered for each case.

The obtained results are shown in Table 4.1. The table presents the characteristics of the best-case and worst-case DP. In addition, there are six more DPs for each of the five resource sets considered. Columns 2 and 5 present the latency results of a traditional list scheduling algorithm. Columns 3 and 6 offer the latency of the solutions produced by the proposed method. Columns 4 and 7 indicate the relative latency improvement of the method as compared to list scheduling. Column 8

presents the relative difference of the latencies for the worst and best case DPs. The last column indicates the execution time of the algorithm. Results show that the proposed method achieves on average a latency improvement of 29% as compared to list scheduling. Also, best-case latency differs on average by about 35% from the worst case. The execution time is reasonably large even for bigger MDGs.

4.4.2 Network Parameter Optimization

The goal of this experiment was to study the performance improvement that results by applying the network optimization step described in the previous section. This step is enabled by the proposed goal-oriented programming model as it supports description of a broad set of constraints on sampling rate, delay, memory, communication, and power consumption. We developed grids of 9, 18, and 27 sensor nodes (SN). Each reconfigurable SN had three modules as follows: one sampling module (Producer-S) with 5 different DPs, one input network module (Producer-N) with 3 DPs, and one output network module (Consumer) with 2 DPs. All DPs represent different latency - power trade-offs. For example, the sampling module includes a sensing frontend block and a data processing block. The DPs of the frontend represent different designs that improve the bit resolution by increasing the sampling rate of $\Delta\Sigma$ ADCs [49, 57], but using more power. The DPs of the processing block are designs for Discrete Fourier Transform (DFT) with different latencies and power consumptions. The assumed Data Aggregation Pattern (DAP) included 8 different paths in the network. The paths provided alternative data communication paths from all nodes in the network to a target point. The optimized cost function included three terms: total power consumption, overall data loss, and size of the required buffers, with the last two being more important. Constraints were minimum data input rate and latency.

The experiments offered insight on how the problem goals and constraints influence the steady-state probabilities of different reconfigurable DPs, hence the nature of the execution adaptation process. We observed that optimal solutions included many DPs with a high power consumption - latency ratio. The data precision constraint is important in selecting the DPs of the sensing component ($Producer - S$). Tightening the power constraint increases the data loss as the optimization algorithm picks the slower DPs for the output network module but these DPs have lower power consumption. The data loss constraint has an important influence on the steady-state

Ex. (1)	Delay Constraint (2)	Total Loss (3)	Max. Loss (4)	Total Power (5)	Total Buffer (6)	Time (sec) (7)
Net 9 (case 1)	4.1	8.85	2.02	4.61	19.90	0.06
Net 9 (case 2)	5.0	7.97	2.18	4.60	3.56	0.06
Net 9 (case 3)	6.5	8.75	1.96	3.65	0	0.06
Net 18 (case 1)	4.4	22.74	5.64	12.14	30.54	0.26
Net 18 (case 2)	5.5	20.33	3.91	11.12	4.47	0.26
Net 18 (case 3)	8.0	18.66	3.91	10.99	1.48	0.26
Net 27 (case 1)	5.6	35.62	3.35	16.94	27.63	0.46
Net 27 (case 2)	6.5	31.50	3.91	15.33	6.42	0.46
Net 27 (case 3)	9.0	29.08	3.91	14.19	3.95	0.46

Table 4.2: Adaptation policy performance characteristics

probabilities of the output network states. Also, increasing the output rate of nodes does not necessarily reduce data loss. Higher output rates might over-constrain the entire path, which can increase the loss of subsequent nodes, and thus lead to higher overall loss and power consumption.

Table 4.2 offers quantitative insight into the experiments. Three latency (delay) constraints, shown in the second column, were considered for each network size. The third column shows the total data loss in the network. The fourth column presents the highest loss at a single node. The fifth column indicates the total power consumption, and the sixth column the total buffer space of the SNs. The last column presents the execution time of solving the linear optimization problem by solver *lp_solve* [11]. The total loss and required buffer space are also relatively low. Specifically, between 14% and 33% of the SNs did not experience any loss. For networks with 27 nodes, between 22% and 50% of the SNs did not require local buffer space to accommodate the rates of their producers and consumers. Power consumption decreases as the latency constraint is relaxed, but the decrease is small due to the smaller weight of power consumption in the cost function.

A second experiment focused on optimizing the DCN parameters of a grid network with 36, 64, 81 and 100 nodes. Each sensing module had 5 reconfigurable DPs, each input network module had 3 DPs, and each output network module had 2 DPs. The adaptation policies were produced for three optimization requirements: minimizing data loss, minimizing average power consumption, and minimizing the total delay. The observed parameters for each policy included the total delay, total data loss, and

Ex.	Max.Data			Min.Loss		
	Delay (msec)	Loss	Power (μW)	Delay (msec)	Loss	Power (μW)
(1)	(2)	(3)	(4)	(5)	(6)	(7)
Netw. 36	9.54	36.13	11.76	12.34	0	10.50
Netw. 64	-	-	-	17.92	5.34	18.80
Netw. 81	15.97	83.09	26.46	22.19	0	23.65
Netw. 100	15.97	100.68	33.04	21.91	0	29.39

Table 4.3: Network performance improvement through adaptation (I)

Ex.	Min.Pow.			Min.Delay			Exec.Time (sec)
	Delay (msec)	Loss	Power (μW)	Delay (msec)	Loss	Power (μW)	
(1)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
Netw. 36	12.44	38.3	9.28	7.22	39.53	19.95	0.703
Netw. 64	18.28	74.73	16.27	11.22	44.39	22.62	2.39
Netw. 81	22.08	91	20.68	13.64	50.46	23.81	3.90
Netw. 100	22.08	112.1	25.58	13.64	89.81	30.43	5.48

Table 4.4: Network performance improvement through adaptation (II)

total power consumption. The results were compared with a greedy policy in which all nodes used their DP with the highest data sensing rate, hence the flexibility of the goal-oriented framework in expressing various constraints is not used.

Tables 4.3 and 4.4 show the obtained results. In table 4.3, Columns 2-4 correspond to the greedy policy, columns 5-7 are for the policy for minimizing loss. In table 4.4, columns 8-10 are for the policy for minimizing average power, and columns 11-13 are for the policy for minimizing the total delay. The columns are labeled in the tables for easy readability. The greedy policy was not satisfying the constraints for the network of 64 nodes: the optimization problem was infeasible as a high input sampling requirement is impossible to meet, if the application demands low loss and the architecture has insufficient buffer memory and communication bandwidth. Column six shows that the data loss is very low, often zero, if the policy is minimized for minimum data loss. However, the data loss of the greedy policy is high, as shown in Column 3. Moreover, the power consumption of the adaptive policy (Column 10) is

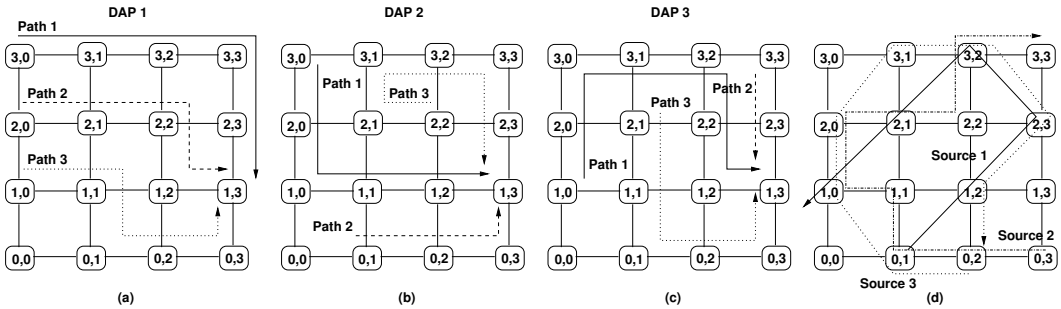

Figure 4.7: Heat source tracking using PSoC network with 16 nodes

around 21% less than for the greedy policy (Column 4). The total delay of the policy optimized for delay (Column 11) is between 14% to 24% lower than the total delay of the greedy policy (Column 2). As compared to the policy optimized for minimum power consumption, the policy for minimum loss increases the total power consumption by up to 13% but significantly reduces the data loss, such as with 112 data values for a network with 100 nodes. The policy with minimum total delay is faster with 37% to 41% than the policy for minimum data loss but its power consumption is with 3% to 47% higher while the data loss is much larger too.

If the sensing rate requirement is increased by 75% then the policy for minimum data loss reduces loss by 61% to 65% as compared to the policy for minimum power consumption, and by 36% to 57% as compared to the policy for minimum delay. However, the policy consumes between 14% to 18% more power than the policy optimized for power and has a total delay between 23% to 30% larger than the policy optimized for minimum total delay. The delay increases because DPs with longer execution time are used more often than the faster DPs but with higher power consumption. Similarly, the loss is higher due to using communication links with lower bandwidth, but which use slower clocks to reduce power consumption.

Column 14 shows the execution time for solving the equations using lp_solve [11]. The solving time is less than 5.5 seconds for models built for networks with 100 nodes. The results show that the optimization methods scales well with the size of the networks.

4.4.3 Heat Source Tracking over Physical Zones

The third experiment tracked a heat source moving over a physical zone. Two zones were considered, one covered by a network of 16 PSoC nodes and one covered by a network of 25 PSoC nodes. The goal was to collect the sensed temperatures at

Ex.	16 nodes					
	Loss			Node delay (msec)		
(1)	≤120 (2)	≤150 (3)	≤180 (4)	<21 (5)	<80 (6)	<138 (max) (7)
DAP 1	1	1	12	3	9	2 (86.74)
DAP 2	1	3	10	3	8	3 (103.45)
DAP 3	1	2	11	3	7	4 (120.16)

Table 4.5: Heat source tracking with static DAPs (16 nodes)

Ex.	25 nodes					
	Loss			Node delay (msec)		
(1)	≤120 (8)	≤150 (9)	≤180 (10)	<40 (11)	<105 (12)	<175 (max) (13)
DAP 1	6	1	16	6	12	5 (136.87)
DAP 2	3	7	13	6	10	7 (170.29)
DAP 3	5	4	14	6	13	4 (170.65)

Table 4.6: Heat source tracking with static DAPs (25 nodes)

the target point with the goal of finding the trajectory of the heat source. Different moving patterns were considered for the heat source, such as the three patterns in Figure 4.7(d) for the network with 16 nodes. Three different DAPs were analyzed, each DAP having three paths. The paths are shown in Figures 4.7(a)-(c). Temperature readings were collected for 3 minutes at a rate of 1 sample per second with a total number of readings of 180 samples per node. The experiments measured the data loss at the nodes, the delay of the sensing nodes, and the delay of the paths.

The effort required to develop the application was relatively small, and depends only marginally on the network size. It is easy to modify a description by updating the goals, events, and constraints. Low and node level descriptions require a higher programming effort as the specific interaction scheme of every node must be provided. Neighborhood level description must be adjusted if the goals and constraints of the application change.

Tables 4.5 and 4.6 summarize the results measured for the two networks, with

Ex.	16 nodes			25 nodes					
	Node delay (msec)			Node delay (msec)					
	(1,0)	(2,0)	(2,2)	(1,0)	(1,1)	(1,3)	(3,1)	(3,3)	(4,0)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
DAP 1	86.92	70.39	36.97	120	103	137	136	36	103
DAP 2	53.68	70.39	103.45	87	70	36.97	136.87	70	137.23
DAP 3	120.16	103.81	36.97	153	170	70.21	70.39	103	103
DAP (15/15/70)	105.92	93.77	46.92	138.15	144.95	75.23	90.23	88	108.13
DAP (15/30/55)	95.21	88.76	56.90	128.25	129.95	70.25	100.17	83.05	113.26

Table 4.7: Heat source tracking with dynamic DAPs

16 nodes and 25 nodes. These measurements are used as a reference to study the effectiveness of dynamically switching between alternative DAPs. The rows correspond to the three static DAPs in Figure 4.7. Columns 2-4 present the number of nodes for which the target point received less than or equal to 120 samples (out of 180 samples), between 121 and 150 samples, and between 151 and 180 samples. Columns 8-10 offer the same information for the network with 25 nodes. Columns 5-7 present the number of nodes that have a delay to the target point less than 21 msec, between 22 and 150 msec, and above 150 msec. The delay of the longest path of a DAP is shown in Column 7 in brackets. Similar information is offered in Columns 11-13 for the network with 25 nodes.

For the network with 16 nodes, the percentage of nodes experiencing data loss is less than 10%. However, for networks with 25 nodes, the percentage is between 30% and 43%. This justifies that data loss becomes significant for larger networks mainly due to more data being transferred through nodes because of longer paths. The overall data loss is 156 values for DAP 1, 190 values for DAP 2, and 206 values for DAP 3. The overall data losses for the network with 25 nodes are 616 values for DAP 1, 655 values for DAP 2, and 856 values for DAP 3. The delay experienced by individual nodes varies significantly depending on the paths topology. Therefore, static paths cannot meet high rate constraints for all nodes, as invariably, few nodes have large delays to the target node. These nodes are the bottlenecks in computing with distributed variables.

Table 4.7 summarizes the results for tracking with dynamic DAPs. The three DAPs were used with two different rate sets: first, DAP 1 with rate 0.15, DAP 2 with 0.15, and DAP 3 with 0.7, and second, DAP 1 with rate 0.15, DAP 2 with 0.3, and DAP 3 with rate 0.55. During execution, DAPs were selected dynamically according

to their rates: each time a new data packet was produced by a node, it was forwarded to the target point along a DAP selected with a probability equal to the rates. The columns indicate the delays experienced by nine sensing nodes when transmitting their data to the target point (each node is denoted by its row and column numbers in the grid). The first three rows are for the static DAPs, and the last two rows for the two dynamic scenarios. Note that the maximum delays are smaller, which reduces the delay for evaluating distributed variables at the target point. The more DAP sets are used dynamically the smaller the difference between a nodes delay and its delay using the shortest path to the target point. Using dynamic DAPs reduced the data loss significantly: only 90 values were lost for the first case, and 109 values were lost in the second case. This is an improvement of 42% and 30% respectively compared to the best case among the static DAPs. For the network with 25 nodes, 550 values were lost for the first dynamic scenario and 600 values for the second dynamic scenario. The first dynamic scenario reduces data loss between 10% and 35% compared to the three static DAPs.

To study the scalability of the methods for larger networks, we developed a SystemC simulation model for the heat tracking application. The model was tuned to give the same data loss and delays as the values measured for the physical network with 25 nodes. For the same static paths and dynamic path configuration rates as in Table 4.7, using DAP 1 with rate 0.15, DAP 2 with 0.15, and DAP 3 with 0.7 reduced data loss on the average by 15% for the network with 36 nodes and by about 10% for the network with 64 nodes. Using DAP 1 with rate 0.15, DAP 2 with 0.3, and DAP 3 with rate 0.55 reduced the loss by about 13% for a network with 36 nodes and around 6% for the network with 64 nodes.

4.5 Conclusions

This chapter describes a performance optimization model that automatically designs the adaptation policies of the reconfigurable sensor nodes to address the goals and constraints of the goal-oriented descriptions. The reconfiguration policies result by finding the switching rates between the design points of the sensing, processing and actuation routines and alternative data communication paths. This is important for co-designing the sensing, processing, actuation and networking subsystems of CPS applications. Adaptation policies are computed by solving a Linear Programming

problem describing adaptation as a Continuous-Time Markov Process. This optimization model targets CPS applications that deal with quasi-static entities. The purpose of this assumption was to thoroughly study the effects of optimization goals and network parameter constraints on resource allocation.

Experimental results present the performance improvement obtained by optimizing the network parameters, and a distributed heat source tracking application using a network of reconfigurable, embedded PSoC processors. Results show that the optimization methods scale well with the network size. The produced adaptation policies offer small data loss, and have low buffer memory requirements. Power consumption can be on average about 21% lower than for policies that are not power aware. Significant data loss results, up to 43%, if a static set of communication paths is used. Static paths are also not effective for computing with distributed variables as the slow nodes become bottlenecks. Switching dynamically between alternative paths reduces data loss by up to 40% and also shortens the delay of the slower nodes. This improves the rates of collecting data at the target points, and helps detecting emergent objects. The execution time of the execution support routines increases linearly with the network size.

Chapter 5

Online Adaptation Policy Design of Parameterized Execution Platform to Track Dynamic Entities[1]

5.1 Introduction

As opposed to the optimization model discussed in Chapter 4 where the purpose was goal-oriented resource allocation to track quasi-static entities, the adaptation policies in this chapter incorporates the additional aspect of the entities being dynamic in nature, i.e. they change their properties over time by moving within the space. The distributed entities are treated as distributed variables, a concept discussed in detail in Chapter 2. In Chapter 4, we discussed the need for switching between Design Points (DPs), that represent different performance-cost trade offs, and data paths depending on goal-oriented descriptions and the resource constraints of the network architecture. Experiments proved that by changing the coefficients in the cost function based on the goal descriptions, the adaptation model managed to meet performance constraints depending on the availability of resources, and sufficient data can be made available for building highly accurate data models. In this chapter, the addition of the dynamic aspect of the entities creates the need for an additional step in the adaptation design that captures the related properties of the dynamic entity, that includes its velocity and direction. Considering different facets of how these

[1] A preliminary version of this work was published in [6]

properties can change over time, trajectory prediction techniques are employed that predict future sensor readings related to the trajectories. These dynamic properties, along with the results of the trajectory prediction algorithms, are utilized by the sensor nodes in their network resource allocation policies to meet the performance requirements of the architecture that would be helpful during the model building procedure.

A typical application that would require this approach of trajectory prediction are target tracking applications, where vehicles that move within a particular region are tracked using acoustic sensors. The research community has published many papers on vehicular tracking, where the purpose is optimizing energy consumption, just like any other work related to Wireless Sensor Networks (WSNs). Zhao [17] optimizes only power where the current sensor, that is tracking a vehicle, predicts the next best sensor that has the highest probability of sampling the vehicle and turns that node on before putting itself to sleep mode. The purpose of trajectory prediction in this research work is incorporating the additional aspects of dynamic properties of the entities and predictions into the adaptation design described in Chapter 4 to build more precise data models that would enable efficient decision making. For example, the schema in Figure 2.1 that specifies the goals of the application described in Figure 1.1 has the output variable *ExpConcentration*, that defines the current and the future (expected) pollution level of a path segment. Based on the current pollution levels and the future predictions, decisions related to vehicular routing are made to minimize overall pollution. The function *pol* in Figure 2.2 defines the pollution level of a car and the procedure can use acoustic sensor readings to determine the driver profile and makes predictions related to the dynamics of the car and the future pollution levels that would be generated from the car. Different enabling conditions can change the velocity and direction of a moving gas cloud, like wind, paths taken by vehicles, temperature, etc. For vehicles, the enabling conditions may correspond to the amount of traffic and the nature of the roads. Hence, apart from ensuring that sufficient amount of data is made available within the application-specific timing constraints for building precise data models, accurately capturing the dynamic properties of the entities and predicting its future positions is equally critical to enable decision making related to application goals.

The research community predominantly uses techniques based on measurement history like Bayesian Inferences [17, 18, 19, 20] and Extended Kalman Filters [21, 22,

23] to make trajectory predictions. This research work considers four facets and/or assumptions including measurement history that might affect the accuracy of trajectory prediction of any moving entity within a region, depending on rate at which the dynamic properties of the entities change and discusses four trajectory prediction methods related to each facet. Experiments were performed on different trajectories where the dynamic properties of each trajectory are different. Conclusions were derived from these experiments and the situations where each algorithm is advantageous over the other algorithms in terms of satisfying performance requirements were identified in the second subsection of the section on experiments. The four algorithms are briefly described below:

- *Quasi-static prediction* based on Bayesian inference where previous readings are used to make predictions using Bayes theorem.
- *Bounded trajectory* method which approximates trajectories as sequences of bounded convex - concave regions and the future trajectory is bounded inside a region that is computed based on the inflexion point that separates the convex-concave regions.
- *Stochastically bounded trajectories*, which also assumes a bounding region for the future trajectory, but describes the transitions from a bounded region to another as a Markov process.
- *An adaptive model*, which performs a dynamic linearization of trajectories by defining states for a trajectory.

The results obtained from these prediction algorithms are used in the adaptation policy design where correlations between the target trajectory and data communication paths are used to improve network performance in terms of reducing overall latency and data loss. Improving latency is critical for generating data models with high precision and accuracy, and improving data loss is important to ensure that sufficient amount of data is available during model parameter generation. The subsequent sections provide the theoretical descriptions of the trajectory prediction algorithms and experimental results provide better insight on the features of each algorithm and the conditions under which each algorithm should be used for better performance in terms of latency and data loss, which are the parameters in the cost function

of the optimization model. The adaptation policy design is devised similar to the one described in Chapter 4 but in this case, the correlations between the selected path and the results from the trajectory prediction algorithm are also considered while switching between design points. The design points include different parameter values corresponding to input buffer size (memory), communication baud rate, and power consumption (radio power levels).

5.2 Related Work

The resources of a network of embedded nodes must be allocated so that there is an optimal load balancing between sensing, processing and communication activities. [77] explain that this problem needs meticulous investigation due to its importance for distributed information processing systems. Several important resource allocation methods have been presented in the literature mainly to optimize energy and power consumption. Munir et al. [65] propose a technique using Markov Decision Processes to tune the parameters of the node architecture (i.e. processor voltage and frequency, and sensing frequency) to reduce energy consumption while operating in changing environments. Software reconfiguration for WSN is discussed in [60]. Lu et al. [78] discuss energy efficient node cluster formation using data correlations and spatial properties of the application. A decentralized adaptive resource allocation approach is discussed in [79]. Nodes use a bidding scheme to allocate resources in which they optimize their return (i.e. the utility of their actions) while minimizing their payments (e.g., consumed energy). Other resource allocation methods for distributed networked systems are discussed in [80, 60, 10, 58]. [17] optimizes power consumption for vehicular tracking applications where each node predicts the next best sensor that has the highest probability of sampling the vehicle and turns that node on before putting itself to sleep mode. While minimizing energy consumption is important to prolong the functioning of a network, acquiring sufficient data is equally important for effective decision making. Performing resource allocation to optimize the quality and quantity of the data acquired through a network has not been studied yet.

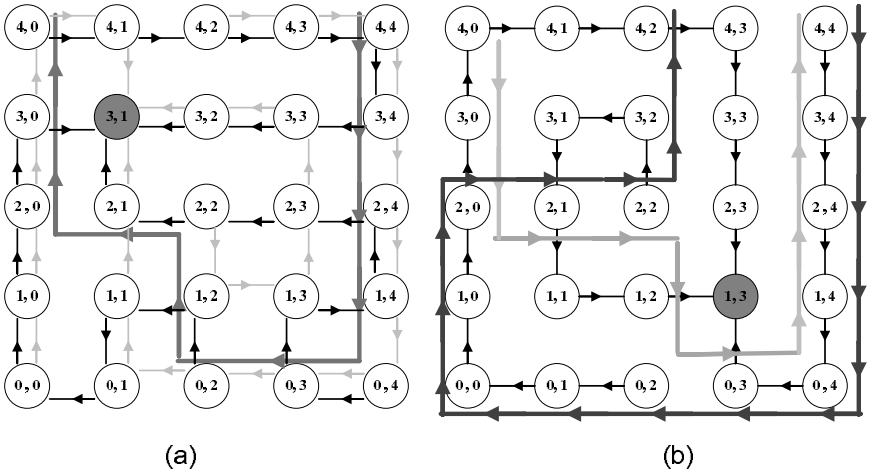

Figure 5.1: Data path configurations used in tracking trajectories

5.3 Motivation

The correlations between the trajectories of the targets and the selected data communication paths influences both the experienced data loss and delays while forwarding data to the target point. The two issues are essential in deciding the quality of the data used to build data model that enable decision making.

In general, data loss occurs when data stored in buffers is overwritten before it is forwarded either because of an ongoing data sampling or data reception. This situation also increases the delay of transmitting data to the target node. For example, Figure 5.1(a) shows two different data path configurations, and the same trajectory (highlighted in bold) runs through the network for those configurations. The target point, which is the node that collects data from all other nodes in the region, is the black bubble. The embedded collection and target points are based on PSoC processor [37]. The first path configuration does not experience any data loss for the considered trajectory. The average delay for the nodes is 1434.62 msec and the maximum and minimum delays are 2040 msec and 1010 msec, respectively. Six nodes in the second path configuration experience data loss for the same trajectory. The average delay for the nodes that did not experience any data loss is 1962.86 msec and the maximum and minimum delays are 5660 msec and 1010 msec, respectively. Hence, nodes have an average delay of 36.82% more than path configuration one.

Figure 5.1(b) shows two different trajectories running through the network which uses the same path configurations, trajectory one is shown with black line and tra-

jectory two with grey line. Trajectory one has data loss at seven nodes compared to trajectory two which has no data loss. The nodes in trajectory one experience an average delay of 73.43% higher than trajectory two. Hence, the same data path configuration can yield different levels of performance for different trajectories.

Using the above experimental example, the need for accurately predicting trajectories of the entities based on its dynamics for efficient model building through optimal path selection to reduce latency and data loss is established.

5.4 Algorithms for Trajectory Prediction

For transmitting data to target points, the data paths that work best are selected for a set of targets. The data paths for data at the receive buffers are observed and also the forwarding path segments that lie in the line of the trajectory. The lengths of these path segments are also considered. The data path that produces the least value of average delay and data loss for the predicted trajectory is chosen. A central element of optimal data path (DAP) selection is predicting the trajectories of the targets.

We first characterize the behavior of a trajectory, and define how the elements in the communication network compute the state of the system based on the information collected. A trajectory is described by its velocity v and angle θ with respect to the X-axis in the grid communication network. The state of a target is defined by its x-position, y-position, and time of triggering an event at a data collection node. The minimum distance between nodes is adjusted according to the range of the wireless communication radios, such that no two nodes can sense a target at the same time t. The variables v and θ correspond to the dynamic properties that define the trajectory state.

Sensing interval for a node is t_s, which depends on processing capacity. Over time t_s, there would be a change in a target's velocity δv which lies in range $[0, \delta v_{max}]$.

$$\delta v_{max} = v_{max} - v \tag{5.1}$$

v_{max} is the maximum velocity.

$$\delta v_{xmax} = \delta v_{max}\ cos\theta \tag{5.2}$$

$$\delta v_{ymax} = \delta v_{max}\ sin\theta \tag{5.3}$$

Over time t_s, there would also be a change in angle $\delta\theta$, which is in range $[0, \delta\theta_{max}]$. $\delta\theta_{max}$ is the worst case change in angle over time t_s, which is computed by using the velocity constraints v_{xmax} and v_{ymax}.

The rate at which the v and θ values change over time i.e. the values of $\delta\theta$ and δv determine how the trajectory moves within the network. Accordingly, there are different facets that describe the dynamics of the trajectory and future trajectory predictions should be made based on these facets. For example, $\delta\theta$ and δv remain constant over time, i.e. the trajectory dynamics change at a constant rate, they are more easily predictable and Bayesian techniques would be able to accurately predict the future trajectory. Similarly, different scenarios, like looping trajectories and trajectories with high fluctuations in velocity can be generated by changing the $\delta\theta$ and δv values over time. Four algorithms were devised for predicting the trajectory of the monitored phenomenon to select the data configuration paths that produce the least data loss and delay while transmitting data from the collection points to the target point for decision making. Each of these algorithms represents a unique feature that tries to capture different scenarios that would change the trajectory dynamics. The first algorithm uses Bayesian inference to make trajectory predictions similar to the sensor selection techniques in [17]. The second algorithm uses bounded trajectories where a Trajectory Approximating Region (TAR) is computed by finding inflexion points of convex-concave fragments. The third algorithm uses stochastically bounded trajectories where the TAR is found based on stochastic techniques. The fourth algorithm is an adaptive algorithm that adapts its prediction based on the current state and the rate of change of parameters related to the trajectory.

5.4.1 Quasi-Static Trajectories

An unknown trajectory of a target is defined to be quasi-static, if the trajectory can be approximated well (at run time) as a collection of fragments pertaining to a set of statically defined trajectories.

Each node stores the static trajectories and the best data communication path for the trajectory. The data path selection procedure dynamically estimates at every

node the most likely static trajectory to which the current node (and its neighboring nodes) might belong to. Then, the optimal data path is dynamically selected for that node. The probability of the current node n being approximated by $traj_{st,i}$ of the static set is estimated using Bayesian inference [17, 81]:

$$p(traj_{st,i}|n) \propto p(n|traj_{st,i}) \sum_{traj_{st,k}} p(traj_{st,i}|traj_{st,k}) p(traj_{st,k}|n_{prev}) \tag{5.4}$$

$p(traj_{st,i}|n)$ is the probability of having trajectory $traj_{st,i}$ given that node n sampled the current trajectory. $p(n|traj_{st,i})$ is the probability of node n sampling data given the static trajectory $traj_{st,i}$. $p(traj_{st,i}|traj_{st,k})$ is the probability of having a transition to $traj_{st,i}$ given $traj_{st,k}$. The probability of having $traj_{st,k}$ given that the previous node n_{prev} sampled the trajectory is $p(traj_{st,k}|n_{prev})$.

The static path having the highest value $p(traj_{st,i}|n)$ is selected as being the one to which the current node belongs to, and its data path is used for forwarding data.

There is a need to build a set of quasi-static trajectories based on the characteristics of the possible trajectories of the targets. Since a trajectory is characterized by velocity v and angle θ, we split equation (5.4) into equations (5.5) and (5.6):

$$p(vel_{st,i}|n) \propto p(n|vel_{st,i}) \sum_{vel_{st,k}} p(vel_{st,i}|vel_{st,k}) p(vel_{st,k}|n_{prev}) \tag{5.5}$$

$p(vel_{st,i}|n)$ is the probability of having velocity $vel_{st,i}$ given that node n sampled the current trajectory. The probability that the velocity of a target sampled at node n is $p(n|vel_{st,i})$. The probability distribution is Gaussian with mean at computed velocity v and variance corresponding to $v_{max} - v$. $p(vel_{st,i}|vel_{st,k})$ is the probability of having a transition to $vel_{st,i}$ given $vel_{st,k}$.

$$p(\phi_{st,i}|n) \propto p(n|\phi_{st,i}) \sum_{\phi_{st,k}} p(\phi_{st,i}|\phi_{st,k}) p(\phi_{st,k}|n_{prev}) \tag{5.6}$$

$p(\phi_{st,i}|n)$ is the probability of having angle $\phi_{st,i}$ given that node n sampled the current trajectory. The probability that the angle of a target sampled at node n is $p(n|\phi_{st,i})$. The probability distribution is Gaussian with mean at computed angle θ and variance corresponding to $\theta_{max} - \theta$. $p(\phi_{st,i}|\phi_{st,k})$ is the probability of having a transition to $\phi_{st,i}$ given $\phi_{st,k}$.

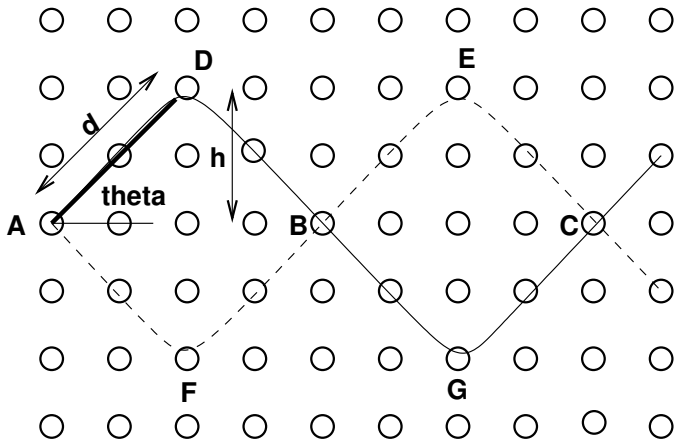

Figure 5.2: Quasi-static trajectory example

The quasi-static trajectories are defined for each node. An example is shown in Figure 5.2. These trajectories look like sinusoidal waves, and their shapes depends on angle θ and the height h, which is determined by computing distance d from velocity v. d is the displacement of the trajectory over time t_s given velocity v.

The sinusoidal shape of the quasi-static trajectories has many advantages. These trajectories can be compressed or expanded by changing angle θ. The height h depends on displacement d over time t_s given velocity v. The sinusoidal shape enables the same trajectory to be reused for multiple nodes and hence reduces programming effort and redundancy. For example, the same trajectory can be defined for all labeled nodes in Figure 5.2. A phase shifted version of the trajectory can also be defined to accommodate all angles. A 90^o phase shifted version of the trajectory is shown in Figure 5.2 (the dotted trajectory). The direction of trajectory motion is also interchangeable. For node B in Figure 5.2, both trajectories together cover angles θ, $\theta + 90$, $\theta + 180$ and $\theta + 270$. Depending on the angle, more phase shifted versions of the same trajectory might be needed to cover all nodes. The smaller the angle, the more the number of phase shifted version of the trajectories are required.

The angle with the highest value $p(\phi_{st,i}|n)$ in equation (5.6) is used as θ. The velocity with the highest value $p(vel_{st,i}|n)$ in equation (5.5) is used to compute d and the segment of that quasi-static trajectory (bolded in the figure for node A) is utilized for prediction. The mapped segment predicts the future sensing nodes of the trajectory. The data communication path is selected by static analysis of that trajectory segment and identifying the data path that performs best for that segment.

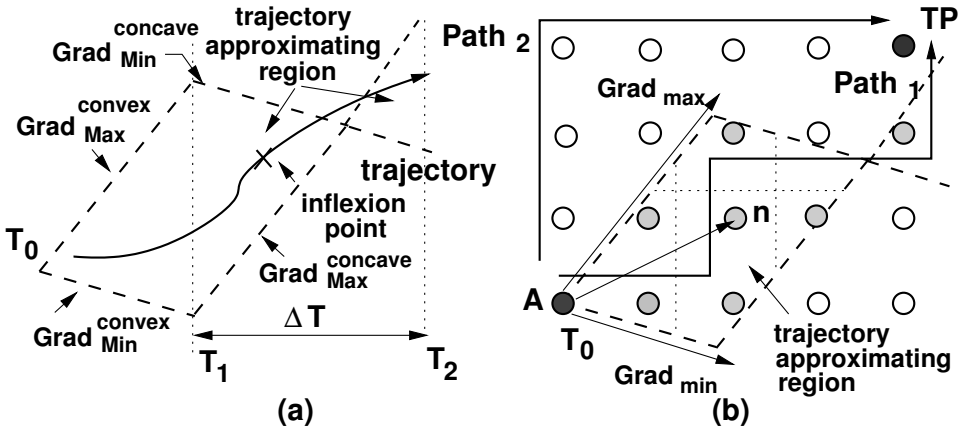

Figure 5.3: Trajectory description using bounded trajectory model

5.4.2 Bounded Trajectories

The second method approximates a trajectory as sequences of convex - concave fragments with bounded gradients of known ranges. The point separating each successive convex - concave fragments is called inflexion point. The average time distance ΔT between inflexion points is known. The prediction model estimates that the real trajectory of a target is located inside the region defined by the minimum and maximum gradients. The region is called Trajectory Approximating Region (TAR).

Figure 5.3(a) illustrates a trajectory expressed in this way. The gradient of the convex fragment is in the range $[Grad_{Min}^{convex}, Grad_{Max}^{convex}]$. At time T_1, there is a break of the two dashed lines as the trajectory switches to the concave part. The gradients of this part are in the range $[Grad_{Min}^{concave}, Grad_{Max}^{concave}]$. The trajectory shown in the figure is approximated well by the corresponding TAR.

The likelihood p_n of node n, inside TAR, to sample the trajectory is estimated as follows. Let's consider a discretization of TAR into sub-regions, as shown with dotted line in Figure 5.3(b). The likelihood p_n depends of the (unknown) length of the trajectory $l_{trajectory}$ inside the sub-region containing the node, the area of the sub-region $Area_{sub-region}$, and the number $N_{alt\ traj}$ of alternative trajectories that are inside the sub-region and go through the node:

$$p_n \propto \frac{l_{trajectory}}{Area_{sub-region}} N_{alt\ traj} \approx \frac{\int_{\Delta T} \sqrt{x(t)^2 + y(t)^2} dt}{Area_{sub-region}} N_{alt\ traj} \tag{5.7}$$

or

$$p_n \propto \frac{\int_{\Delta T} \sqrt{x_0^2 + y_0^2 + 2(Grad_x + Grad_y)t}\ dt}{Area_{sub-region}}\ N_{alt\ traj} \tag{5.8}$$

$x(t)$ and $y(t)$ are the (unknown) equations describing the target's trajectory. x_0 and y_0 are the coordinates of the target at time T_0, and $Grad_x$ and $Grad_y$ are the gradients at time T_0 of the trajectory along the two axes.

The number of alternative trajectories $N_{alt\ traj}$ can be estimated based on (i) the angle defined by vector $\overrightarrow{(A,n)}$ from node A (the node currently considered) to node n and the vector corresponding to $Grad_{min}$, and (ii) the angle defined by vector $\overrightarrow{(A,n)}$ and the vector corresponding to $Grad_{max}$. The larger the product of the two angles the higher $N_{alt\ traj}$:

$$N_{alt\ traj} \propto \overrightarrow{(A,n)}, \widehat{\overrightarrow{Grad_{min}}} \times \overrightarrow{(A,n)}, \widehat{\overrightarrow{Grad_{max}}} \tag{5.9}$$

To reduce the overhead of estimating likelihood p_n, starting from expressions (5.8) and (5.9), two approximations can be introduced to calculate probability p_n.

The first approximation considers that all nodes of a TAR are equally likely to be part of the trajectory, therefore expression (5.8) has the same value for all nodes, and is computed using the area value for the entire TAR, without any discretization. N_{alt} is the same for all nodes and can be eliminated from the expression. This approximation reduces the computing overhead to estimate p_n at node A.

The second approximation reduces the overhead even further assuming that a target's trajectory can have any gradient inside TAR, hence, in the worst case, it coincides with the data communication path inside the region. Then, the likelihood p_n can be estimated as follows:

$$p_n \propto \frac{1}{length_{n \in DP_i}} \tag{5.10}$$

DP_i is the data communication path containing node n. $length_{n \in DP_i}$ is the length of DP_i inside TAR. For example, in Figure 5.3(b), data communication $Path_2$ has no nodes inside the regions, hence the likelihood of its nodes sampling the trajectory is lesser than for the nodes of $Path_1$. Counting the number of path nodes inside a TAR requires low processing overhead. Note that for windows of size one, expres-

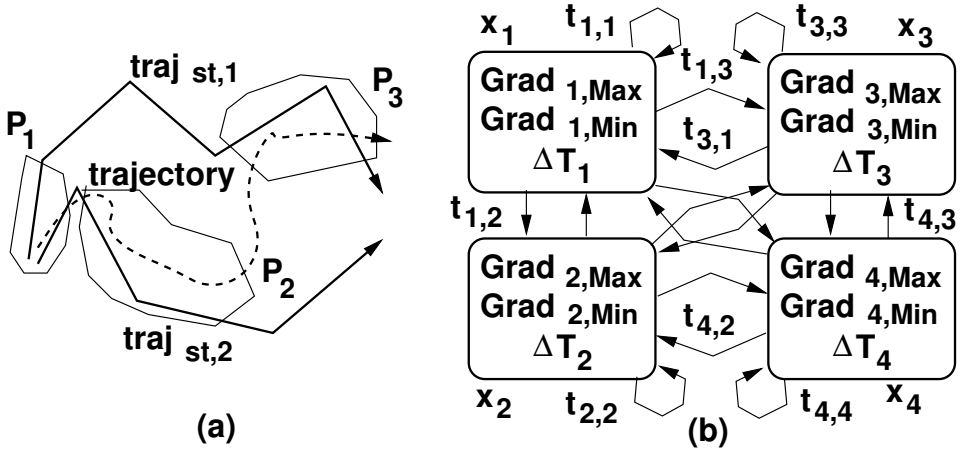

Figure 5.4: Trajectory description using stochastically bounded trajectory model

sion (5.8) is proportional to the cosine of the angle between the trajectory and the data communication path at node A.

5.4.3 Stochastically Bounded Trajectories

The third model represents a trajectory as stochastically bounded trajectories, if the trajectory is approximated by bounded fragments in which the change to another fragment follows a stochastic rule, instead of alternating between convex and concave regions like for bounded trajectories. Similar to bounded trajectories, stochastically bounded trajectories are composed of fragments with gradients limited to a known range $[Grad^i_{Min}, Grad^i_{Max}]$. Figure 5.4(b) shows the description of a trajectory that is composed of four fragments, each being characterized by a specific gradient ranges and average time ΔT of switching to another fragment.

The switching between different fragments is modeled as a Markovian process. Lets denote x_i the steady-state probability of fragment i, and $t_{i,j}$ the transition rates between the fragments. The values of $t_{i,j}$ are found through observing various trajectories through the same region. Then, the steady-state probabilities can be computed by solving the following equations:

$$t_{i,i}x_i - \sum_{j \neq i} t_{j,i}x_j = 0 \tag{5.11}$$

and

$$\sum_{\forall k} x_k = 1 \tag{5.12}$$

Using the same reasoning as for bounded trajectories, the likelihood p_n in expression (5.8) is updated as follows:

$$p_n \propto x_i \ \frac{\int_{\Delta T} \sqrt{x_0^2 + y_0^2 + 2(Grad_x + Grad_y)t}\ dt}{Area_{sub-region}}\ N_{alt\ traj} \tag{5.13}$$

or the approximation in equation (5.10) is changed to expression:

$$p_n \propto x_i \ \frac{1}{length_{n \in DP_i}} \tag{5.14}$$

5.4.4 Adaptive Model

The fourth trajectory prediction method adapts its prediction based on the current state and the rate of change of parameters related to the trajectory. The method incorporates the advantages of the first three methods. The state of a trajectory depends on the rate at which its parameters change. If the rate of change of velocity or angle changes beyond a range, the trajectory has entered another state. For example, a higher rate of change of angle causes the trajectory to loop. A lower rate of change of angle causes an almost linear trajectory.

The proposed mechanism utilizes this definition of a trajectory state to make the following predictions: If the rate of change of angle and velocity over time has been constant (e.g., acceptable increase or decrease of ϕ for angle, and acceptable increase or decrease of $x\frac{m}{s}$ for velocity, to introduce noise margin), the trajectory is assumed to remain in the same state. A new state is defined by change of velocity ΔV and angle $\Delta\theta$ over time. As shown in equation (5.15), depending on the time for which the trajectory has been in the above state, we can compute the probability p_{state} for which it remains in the same state. That depends on the number of previous events for which the trajectory assumed the same state $traj_{state_n}$ and the deviation of current values of velocity $\delta(\Delta V)$ and angle $\delta(\Delta\theta)$ with respect to the current state. Hence,

$$p_{state} \propto \frac{traj_{state_n}}{\delta(\Delta V).\delta(\Delta\theta)}. \tag{5.15}$$

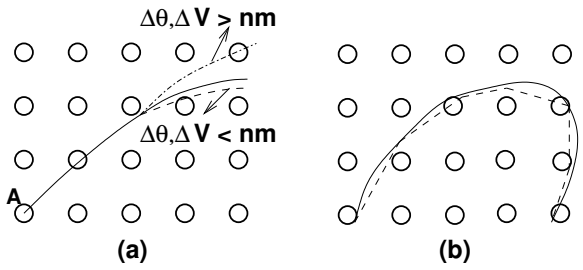

Figure 5.5: Adaptive prediction model: (a) noise margins and (b) linear approximation

For higher probabilities, we can make predictions for longer future trajectory segments, which depends on time $t_{predict}$ over which the trajectory can be predicted accurately.

$$t_{predict} \propto p_{state} \tag{5.16}$$

Hence,

$$t_{predict} \propto \frac{traj_{state_n}}{\delta(\Delta V).\delta(\Delta\theta)} \approx k.traj_{state_n}.(x - \Delta V).(\phi - \Delta\theta) \tag{5.17}$$

The higher the distance of ΔV and $\Delta\theta$ from the noise margins, the closer it is to the trajectory state parameters and hence the probability or time of prediction increases. The above equation is valid only when $x > \Delta V$ and $\phi > \Delta\theta$. Otherwise, the trajectory has entered a new state: $t_{predict}$ is updated by changing $traj_{state_n}$ to value one, and a new state is defined corresponding to the change in angle and velocity with respect to the previous node.

$t_{predict}$ determines the number of future events that can be predicted. This depends on the current state of the trajectory, the current estimated position of the target, and the current absolute values of angle and velocity. When the trajectory enters a new state, $traj_{state_n}$ is reset to value one, and the value of $t_{predict}$ is determined by the proportionality constant k.

The noise margins ϕ for angles and x for velocity are defined as follows. If the rate of change of parameters go beyond these parameters, the trajectories change states. Refer to Figure 5.5(a), where the solid line is the trajectory if ΔV and $\Delta\theta$ do not

change over time from node A. The broken line is the deviation that results in the same state since it triggers the same set of future nodes, but the dotted and broken line triggers a different set of nodes and hence belongs to a different state. In this case, the change in parameters exceeded the noise margins. The two noise margins are defined as follows:

$$\delta(\Delta\theta) \leq \phi \tag{5.18}$$

and

$$\delta(\Delta V) \leq x \tag{5.19}$$

We use the computed value of $t_{predict}$ to find parameters ϕ and x. We then compute the future trajectory using $t_{predict}$ assuming that the rate of change of parameters is equal to the values defined for the state. Using the above information, we determine future sensing nodes for that state. The minimum change in velocity and angle, that generates a trajectory producing a different set of future sensing nodes, correspond to the noise margin.

Moving on to the prediction procedure, for $t_{predict}$ seconds, the future trajectory is approximated based on current velocity, angle, and trajectory state (rate of change of velocity and angle) using linear approxation. The method produces trajectory segments that approximate the trajectory curve. The curve in Figure 5.5(b) shows the trajectory and the broken lines present the segments produced by the linear approximation method. Linear approximation is a good method for prediction in this case since we do not have to reproduce the actual trajectory, but only determine the future sensing / processing nodes.

Value $t_{predict}$ depends on the trajectory state. If the trajectory remains in the state for some time, value $t_{predict}$ is high and a large segment of the future trajectory can be predicted. The value of $t_{predict}$ is low at the point where the trajectory state switches. Hence, a very small segment of the trajectory is predicted until the trajectory settles into a new state.

5.5 Parameter Optimization using Predictions

This section describes adaptation policies to select the parameters of the nodes during communication along data paths, namely buffer size, baud rate and radio power depending on the available resources and trajectory prediction. In this section, we contend that apart from optimizing data paths, there is also scope for improving performance by adjusting these SN-level parameters based on trajectory predictions. Let us consider the example shown in Figure 5.6 where node n selects the path drawn in the figure and the dotted curve is the future trajectory. It can be observed that a major section of the forwarding nodes in a selected data path lies in the vicinity of the future trajectory. In this case, the forwarding nodes would have to store data from node n in the input buffers while processing the sensed data corresponding to the target. This would result in longer waiting times at the input buffers of the forwarding nodes, increasing latencies and also the risk of buffer overwrites resulting in data loss. Data loss can be prevented by increasing the input buffer sizes of each forwarding node. Data loss can also be prevented by adjusting the baud rate to a value relative to the speed of the trajectory and this prevents longer waiting times at the input buffers, lowering the risk of buffer overwrites. It was observed that lower baud rates actually improves data loss without affecting latencies as the time spent by data at the input buffers of the forwarding nodes is now spent in communication.

The subsequent subsections formulates adaptation policies that uses the above concept where results from trajectory prediction are used by the nodes to switch between network parameters to minimize the cost function by reducing data loss and average delay.

5.5.1 Optimizing Buffer Size

The following constraints refer to buffer size:

$$N_n^{in} \leq B_{fk} \tag{5.20}$$

and

$$B_{fk} \leq Buff_{cap} \tag{5.21}$$

N_n^{in} is the amount of data input at a node, B_{fk} is buffer size, which ranges from one to x (e.g., $x = 4$ bytes in our practical experiments), and $Buff_{cap}$ is the buffer capacity at the node, which depends on the amount of available memory. The larger the amount of memory required for processing, the lower is the buffer capacity. Hence, we use four DPs in our experiments corresponding to buffer size.

It was observed that higher the buffer size, higher is the average delay. For lower buffer sizes, data gets overwritten, and hence does not contribute to average delay since it is computed only for nodes that reach the target point.

$$Average\ Delay = \frac{\sum loss\ of\ individual\ nodes}{\sum No.\ of\ nodes\ recvd.\ at\ TP} \tag{5.22}$$

The performance equation for buffer size is as follows:

$$-P_{Bf} = \alpha_{Bf} Loss + (1 - \alpha_{Bf}) Delay \tag{5.23}$$

The negative sign shows that performance improves when we reduce loss and delay. A large value of α_{Bf} indicates that the focus is to improve delay over loss. This decision can be made based on the current state of the trajectory, the length of the critical path ($l(Cr_p)$) from current node to the target point (TP), and the length of the future trajectory in the bounding region that covers the area between the current node and the TP, as shown in Figure 5.6. If the length of the critical path is such that the projected delay is less than the threshold value that would make the data obsolete, the focus could be more on improving data loss. At the same time, sufficient amount of data should reach TP for efficient decision making.

$$l(Cr_p) = max(l(p_1), l(p_2), ..., l(p_n)) \tag{5.24}$$

The value of α_{Bf} depends on the following two parameters:

$$\alpha_{Bf} \propto \frac{l(Cr_p)}{l(traj_{bound})} \tag{5.25}$$

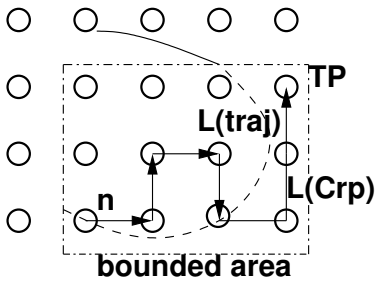

Figure 5.6: Parameter optimization using predicted trajectory

A large critical path results in longer delay if buffer size is higher and if a major part of the future trajectory lies inside the bounded area. This increases the possibility of data loss due to data overwrites during processing at the future nodes. α_{Bf} is directly proportional to the length of the critical path, and inversely proportional to the length of the trajectory bounded by the area. The computed α_{Bf} is used to compute the steady state probabilities in equations (5.26) and (5.27):

$$t(i,i)x_i - \sum_{\forall k \in Bf} t(k,i)x_k = 0 \tag{5.26}$$

where, $t(i,j)$ is the transition probability from state (DP) i to j, and x_i-s are the steady state probabilities.

$$\sum x_i = 1 \tag{5.27}$$

In our case, the number of elements, k is four.

5.5.2 Optimizing Baud Rate

Higher baud rates result in faster communication between neighboring nodes which lowers delay. However, slower baud rates, in certain conditions enumerated below, can help improve performance:

1. Improving power efficiency to prolong network life: The default baud rate is switched to a lower value to conserve battery life.

2. Preventing data loss by ensuring shorter waiting times at the input buffers of the processing nodes, thus lowering the risk of a data overwrite condition: The

baud rate is lowered to a value relative to the speed of the trajectory such that data is forwarded at a speed slower than the trajectory. We can avoid long waiting times at input buffers using this method but the final average delay must be comparable to the average delay with the previous baud rate and the delay should not exceed the threshold value that makes that data obsolete.

The following is the baud rate performance equation based on the above conditions:

$$-P_{Br} = \alpha_{Br} Loss + \beta_{Br} Delay + \gamma_{Br} Power \tag{5.28}$$

The coefficient α_{Br} is computed based on the length of the projected trajectory ($l(traj)$) within the bounded area shown in Figure 5.6:

$$\alpha_{Br} \propto \frac{1}{l(traj_{bound})} \tag{5.29}$$

The coefficient β_{Br} is computed based on the delay of path p_i given a baud rate value. The constraint for computing this parameter is that the delay should be less than the threshold value that makes this data obsolete at TP. In equation (5.30), β_{Br_k} finds the coefficient for each bandwidth value and the selected path p_i. The baud rate that performs best is used as β_{Br} in equation (5.28).

$$\beta_{Br_k} \propto \frac{1}{Delay_{pi_{Br}}} \tag{5.30}$$

and

$$Delay_{pi_{Br}} = l(p_i).B_r + t_{Br_{wait}} \tag{5.31}$$

and

$$t_{Br_{wait}} = \sum_{\forall n \in pi} t_{tn} - t_{rn} \tag{5.32}$$

where t_{rn} is the time at which data is received at node n and t_{tn} is the time at which data from current node is transmitted from node n. $t_{Br_{wait}}$ corresponds to the waiting times at the input receive buffers of the nodes. If the forwarding node is

predicted to be performing the task of processing data, the current node can reduce its bandwidth to an order of magnitude corresponding to the node processing time. This results in lower values of $t_{Br_{wait}}$ and reduces the risk of a data overwrite.

In our experiments, we have 3 DPs for baud rate at the SN-level (communication between embedded node and radio module using SPI protocol), with bandwidth values equal to 4.8kbps, 9.6kpbs and 19.2kbps, where 9.6kbps is the default value selected by the nodes. We have 2 DPs for baud rate at the network level (inter-node communication using radio modules), with bandwidth values equal to 31.25kbps and 62.5kbps. The default value is 31.25kbps.

5.5.3 Optimizing Power Consumption

The last coefficient depends on the amount of power available for data communication modules and the desired length of battery life:

$$\gamma_{Br} \propto \frac{Poweravailable(watts)}{l(Battery_{desired})} \tag{5.33}$$

The computed values of α_{Br},β_{Br} and γ_{Br} are normalized to compute the weights for equation (5.28). These weights are used to decide the appropriate baud rate.

Power consumption is adjusted using 8 DPs that correspond to the radio power levels that range from 0 to 7 with 7 being the highest power level that minimizes the number of re-transmissions and the default power level is 1.

5.6 Experiments

Four trajectory prediction algorithms were studied using eight trajectories defined such that they cover the entire data communication network. Trajectories one to five cover different possible trajectory directions, velocity values, angle ranges, and rates of change of velocity and angle. These parameters are important for the prediction performance of the algorithms. Situations where the algorithms perform worst were identified and then trajectory six was built based on these situations. Hence, all algorithms are expected to yield less optimal results on trajectory six. Figure 5.7 illustrates the first six trajectories. Finally, we defined two more trajectories that are approximately five to six times longer than trajectories one to five and three

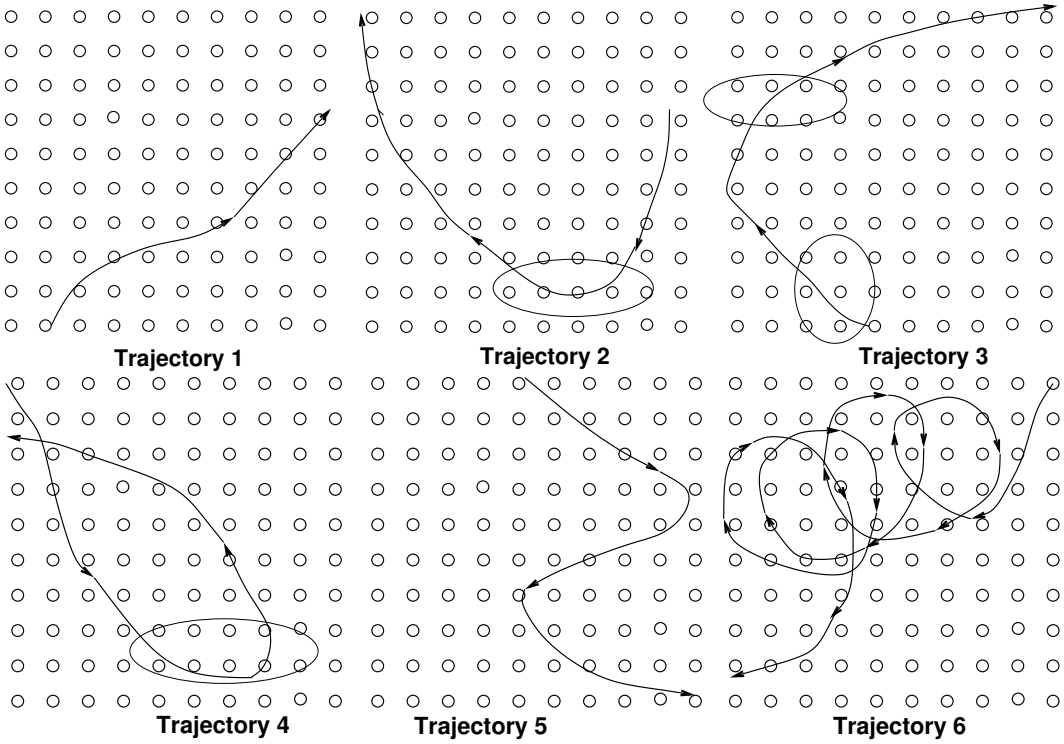

Figure 5.7: Trajectory descriptions

times longer than trajectory six and we named them trajectory A and trajectory B. We considered trajectories four and six, which have the most number of events, to create trajectories A and B that move along the same direction as that of trajectories four and six while looping along its current position. To study multi-target tracking, the experiments also considered multiple trajectories simultaneously. The data paths configurations were defined such that there are three unique path configurations for each of the four target points. Hence, the experiments used a total of twelve path configurations.

The experiments section is divided into three subsections. The first subsection provides experimental results and analysis of the four trajectory prediction algorithms on the eight trajectories. The second subsection provides conclusions derived from the experimental analysis detailed in Subsection 4.6.1 and describes the conditions under which each of the four algorithms would perform optimally by making accurate trajectory predictions. The third subsection provides experimental results for adaptation policy design where design points for parameters including buffer size, baud rate and power consumption were switched based on the trajectory predictions. The

Table 5.1: Performance for Trajectory 1

TP	**Path**	Data Loss (20 events)					Average Delay				
	Conf.	**Static**	**A1**	**A2**	**A3**	**A4**	**Static**	**A1**	**A2**	**A3**	**A4**
1	2	3	4	5	6	7	8	9	10	11	12
1	1	0					947				
	2	4	1	1	0	0	509	259	298	273	188
	3	2	(-1)	(-1)	(0)	(0)	251	(-3%)	(-18%)	(-9%)	(25%)
2	1	1					647				
	2	3	0	1	0	0	752	325	376	383	267
	3	1	(1)	(0)	(1)	(1)	416	(22%)	(10%)	(8%)	(36%)
3	1	8					644				
	2	5	1	1	0	1	715	240	326	300	224
	3	1	(0)	(0)	(1)	(0)	323	(26%)	(0%)	(7%)	(31%)
4	1	4					780				
	2	2	2	2	1	0	716	456	399	586	366
	3	2	(0)	(0)	(1)	(2)	417	(-9%)	(4%)	(-40%)	(12%)

parameter optimization technique was tested on trajectories A and B.

5.6.1 Experimental analysis of the four prediction algorithms

The tables presenting the experimental results have the following structure described below. Columns one and two indicate the static path configurations used for each target point. Columns three to seven show data loss for different configurations and the number of events generated by the trajectory (bracketed number in row one). Columns eight to twelve show average delay for different configurations. Specifically, Column three shows the data loss and column eight indicates the average delay for static path configurations. Column four gives the data loss performance and Column nine presents the average delay performance for prediction using quasi-static trajectories. Column five shows the data loss performance and Column ten indicates the average delay of trajectory prediction through bounded trajectories. Column six shows the data loss performance and Column eleven the average delay for using stochastically bounded trajectories. Column seven presents the data loss and Column twelve the average delay of the adaptive algorithm. Columns eight to twelve also show the percentage improvement of average delay (the percentage values in brackets below the absolute average delay values) with respect to the best case path configuration. Columns three to seven show the absolute improvement in data loss with respect to the best case path configuration.

Table 5.2: Performance for Trajectory 2

TP	**Path**	Data Loss (21 events)					Average Delay				
	Conf.	**Static**	**A1**	**A2**	**A3**	**A4**	**Static**	**A1**	**A2**	**A3**	**A4**
1	2	3	4	5	6	7	8	9	10	11	12
1	1	6					1134				
	2	5	0	0	0	0	868	298	308	305	288
	3	0	(0)	(0)	(0)	(0)	308	(3%)	(0%)	(1%)	(7%)
2	1	4					925				
	2	2	0	2	0	0	748	364	448	387	333
	3	2	(2)	(0)	(2)	(2)	448	(19%)	(0%)	(14%)	(26%)
3	1	8					1054				
	2	1	1	1	0	0	844	346	360	383	317
	3	1	(0)	(0)	(1)	(1)	360	(4%)	(0%)	(-6%)	(12%)
4	1	5					799				
	2	5	0	1	0	0	871	319	312	331	203
	3	1	(1)	(0)	(1)	(1)	312	(-2%)	(0%)	(-6%)	(35%)

Trajectory 1. Refer to Table 5.1 for the performance of trajectory one. This is the simplest trajectory of all cases.

All algorithms make good predictions for this trajectory. The adaptive method (A4) gives the best improvement in average delay compared to all other algorithms for all four TPs. The improvement ranges from 12% to 36%. The method is also the only algorithm that produces a positive improvement in average delay for all four target points. Even though quasi-static prediction (A1) produces good predictions, it is unable to produce improvements as significant as method A4 as its path selection procedure depends on the performance of the mapped quasi-static trajectory segments. The algorithms perform worst when using TP one for this trajectory as three out of four algorithms produce negative improvements in average delay, and two out of four algorithms produce a negative improvement in data loss.

Trajectory 2. Refer to Table 5.2 for the performance of this trajectory. This trajectory assumes a motion that initially runs at high speed, i.e. in the range of 65% to 75% of v_{max} until it slows down to 40% to 50% of v_{max} to curve at the segment circled in Figure 5.7 before it speeds up again. This trajectory traverses through the entire network but the rate of change of angle is uniform. The slow speed at the circled segment gives enough time for the trajectory to curve.

There are only insignificant mispredictions by the quasi-static algorithm (A1) at a couple of nodes in the circled area. The predictions by bounded trajectory algorithm

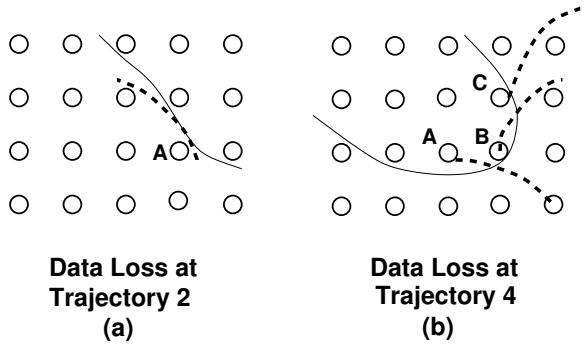

Figure 5.8: Data loss at trajectories 2 and 4

(A2) resulted in producing the same performance as the best case path configuration for all target points. Hence, improvements in data loss and average delay by algorithm A2 for all TPs is zero. All other algorithms give good improvements in data loss. The adaptive algorithm (A4) gives the best improvement in average delay for all four TPs ranging from 7% to 35%. The stochastically bounded algorithm (A3) produces good improvements in data loss but generates negative improvements in average delay for two out of four TPs. At TP three, algorithm one loses data due to prediction inaccuracy. The scenario is discussed with the help of Figure 5.8(a). Node A in the figure is the one whose data is lost. The continuous curve represents the trajectory and the broken curve is the quasi-static trajectory segment that node A mapped.

Trajectory 3. Refer to Table 5.3 for the performance of this trajectory. This trajectory was defined to cover the area of the network that was uncovered by the first two trajectories. Trajectory three might look symmetrical to trajectory two but the dynamics is completely different. Velocity values change in this trajectory over a wide range from 20% to 96% of v_{max}, but the rate of change of velocity over time for most of the trajectory is less than the threshold for a grid distance of 3m. The other trajectories do not have this wide velocity range.

All algorithms give good improvement over data loss and average delay. The data loss produced by the algorithms when target point three is used is unavoidable and is not due to mispredictions. The adaptive algorithm (A4) gives the best improvement in average delay for all target points and the characteristics of two trajectory segments (circled in Figure 5.7) were identified to be the reason for this behavior. The Trajectory Approximating Regions (TARs) estimated by bounded trajectory (A2) and statistically bounded (A3) methods are sufficient to avoid data loss, but not even 50%

Table 5.3: Performance for Trajectory 3

TP	**Path**	Data Loss (25 events)					Average Delay				
	Conf.	**Static**	**A1**	**A2**	**A3**	**A4**	**Static**	**A1**	**A2**	**A3**	**A4**
1	2	3	4	5	6	7	8	9	10	11	12
1	1	4					830				
	2	5	1	2	0	0	1019	385	407	417	314
	3	2	(1)	(0)	(2)	(2)	407	(6%)	(0%)	(-2%)	(23%)
2	1	7					985				
	2	7	0	1	0	0	841	284	292	317	242
	3	1	(1)	(0)	(1)	(1)	292	(3%)	(0%)	(-8%)	(17%)
3	1	6					1052				
	2	3	1	2	1	1	680	379	425	410	319
	3	2	(1)	(0)	(1)	(1)	474	(20%)	(10%)	(14%)	(33%)
4	1	8					865				
	2	0	0	1	0	0	684	299	341	321	293
	3	1	(0)	(-1)	(0)	(0)	342	(13%)	(0%)	(6%)	(14%)

of the area of TARs generated by the sampling nodes contain the actual trajectory. This results in higher average delay. This is the reason for negative or insignificant improvements of average delay. Algorithm A4 gives the best improvement in average delay ranging from 17% to 33%. Algorithms A3 and A4 give the best improvement in data loss.

Trajectory 4. Refer to Table 5.4 for the performance of this trajectory. This trajectory starts going down the network and then loops back at the circled area in Figure 5.7. At that area, the rate of change of velocity is uniform but the rate of change of angle changes abruptly causing quasi-static algorithm (A1) to make mispredictions shown in Figure 5.8(b). Nodes A, B and C map to segments (dotted curves) that completely mispredict the trajectory resulting in data loss. TPs one and two provide a good reflection of this misprediction and the three nodes that lose data are nodes A, B and C. Bounded trajectory algorithm (A2) performs similar to the best case path configuration. The other three algorithms give good improvement in data loss with adaptive algorithm (A4) giving the best improvement. Algorithm A4, again, gives the best improvement in average delay compared to the other algorithms. All algorithms give poor improvements in average delay when TP two is used. Algorithms A1 and A3 give negative improvements and algorithms A2 and A4 give an improvement of 0%.

Trajectory 5. Refer to Table 5.5 for the performance of this trajectory. Algorithm

Table 5.4: Performance for Trajectory 4

TP	Path	Data Loss (30 events)					Average Delay				
	Conf.	Static	A1	A2	A3	A4	Static	A1	A2	A3	A4
1	2	3	4	5	6	7	8	9	10	11	12
1	1	10					1007				
	2	9	3	6	4	1	1133	478	570	452	432
	3	6	(3)	(0)	(2)	(5)	550	(13%)	(-3%)	(18%)	(21%)
2	1	5					887				
	2	8	3	5	0	0	834	452	300	425	299
	3	5	(2)	(0)	(5)	(5)	298	(-52%)	(0%)	(-43%)	(0%)
3	1	5					733				
	2	13	0	4	0	1	902	356	426	264	257
	3	4	(4)	(0)	(4)	(3)	425	(16%)	(0%)	(38%)	(40%)
4	1	8					941				
	2	9	0	4	0	0	925	258	266	279	249
	3	5	(5)	(1)	(5)	(5)	285	(10%)	(7%)	(2%)	(13%)

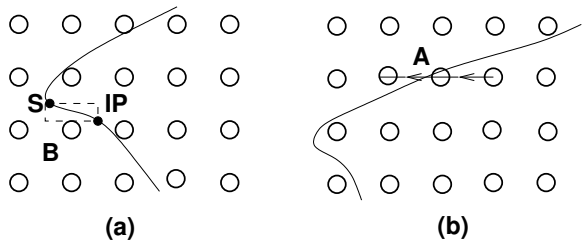

Figure 5.9: Data loss at trajectory 5

A2 gives no improvement in data loss, while the other three algorithms give good improvements. Algorithm A4 gives the best improvement in average delay, which is above 25% except for TP two for which all algorithms give negative or insignificant improvements.

This trajectory yields the following two interesting aspects of the network behavior:

1. When TP one is chosen, node A in Figure 5.9(b) loses data because all three path configurations for TP one map to the same path segment (dotted lines in the figure) that results in data getting overwritten at the last node in the segment. All three algorithms made accurate predictions and still, data was lost at node A. This is a situation that is beyond the three algorithms and a good topic for future work where a method could be devised to switch between TPs by reducing switching overhead.

Table 5.5: Performance for Trajectory 5

TP	Path	Data Loss (26 events)					Average Delay				
	Conf.	Static	A1	A2	A3	A4	Static	A1	A2	A3	A4
1	2	3	4	5	6	7	8	9	10	11	12
1	1	2					800				
	2	2	1	2	1	1	807	410	444	393	338
	3	2	(1)	(0)	(1)	(1)	444	(8%)	(0%)	(12%)	(24%)
2	1	6					883				
	2	1	1	1	0	0	761	290	293	412	287
	3	1	(0)	(0)	(1)	(1)	293	(1%)	(0%)	(-41%)	(2%)
3	1	8					740				
	2	7	0	1	0	0	960	257	293	364	209
	3	1	(1)	(0)	(1)	(1)	292	(12%)	(0%)	(-25%)	(28%)
4	1	5					727				
	2	6	0	2	0	0	809	240	309	212	215
	3	2	(2)	(0)	(2)	(2)	271	(11%)	(-14%)	(22%)	(21%)

2. Referring to Figure 5.9(a), algorithm two made the right prediction at node B by computing the inflexion point (labeled IP) accurately. The point labeled S is the sampling point of the node. But the TAR proved to be too small for node B to make a path selection. It ended up selecting the default path which resulted in data loss. This situation occurs when IP is computed by the node at a location that is too close to the node. If the algorithm can get at least the next sampling node in its TAR, it will be able to avoid data loss.

Trajectory 6. Refer to Table 5.6 for the performance of this trajectory. The target in this trajectory loops while moving in a linear direction. Algorithms A1 and A2 give poor improvements over data loss, but algorithms A3 and A4 give good improvements. Algorithm A1 gives insignificant improvement in average delay, algorithms A2 and A3 give negative or no improvement in average delay, and algorithm A4 gives significant improvements in average delay for all target points, excluding TP two.

Algorithm A1 produces wrong predictions for most of the nodes at places where the trajectory loops. It is able to produce fairly accurate predictions at the linear part of the trajectory (i.e. the trajectory parts that are bolded in Figure 5.10(a)). The looping part of the trajectory was selected for analysis because any segment of this trajectory that loops similar to Figure 5.10(b) produces similar mispredictions. The broken lines in the figure shows the quasi-static trajectory mapping of the nodes.

It was observed that the TARs produced by algorithm A2 are very small at the

Table 5.6: Performance for Trajectory 6

TP	Path	Data Loss (74 events)					Average Delay				
	Conf.	Static	A1	A2	A3	A4	Static	A1	A2	A3	A4
1	2	3	4	5	6	7	8	9	10	11	12
1	1	23					803				
	2	10	5	7	1	0	912	305	334	414	296
	3	7	(2)	(0)	(6)	(7)	328	(7%)	(-2%)	(-26%)	(10%)
2	1	33					877				
	2	8	10	7	4	2	772	398	356	377	347
	3	8	(-2)	(1)	(4)	(6)	350	(-14%)	(-2%)	(-8%)	(1%)
3	1	22					953				
	2	18	7	8	3	2	840	315	323	389	285
	3	8	(1)	(0)	(5)	(6)	323	(3%)	(0%)	(-20%)	(12%)
4	1	13					842				
	2	24	11	13	4	2	1048	473	515	584	378
	3	14	(3)	(1)	(10)	(12)	517	(9%)	(0%)	(-13%)	(27%)

areas where the trajectory loops reducing path selection efficiency even when prediction accuracy is high. This results in poor improvement over data loss. In three cases, A2 produces a TAR similar to Figure 5.9(a) resulting in data loss. At the areas where the trajectory is linear, i.e. the bolded part in Figure 5.10(a), the area produced is large but does not cover much of the trajectory. In other words, the IP computed is inaccurate especially in those areas where the trajectory just stopped looping and the linear part of the trajectory starts.

Algorithm A3 produces TARs with larger area but the TARs cover only a small part of the future trajectory. This condition is sufficient to give good improvements over data loss but results in increasing the average delays for all four TPs.

Algorithm A4 is able to produce accurate predictions resulting in the best improvement in both data loss and average delay for all TPs. It produces low values of $t_{predict}$ at places where the trajectory loops, but it is able to map to the appropriate paths efficiently.

Multiple trajectories. The execution time for simultaneous prediction of three trajectories is 2500 msec on an average, 2650m sec on an average for six trajectories, and 2700 msec on an average for nine trajectories. Table 5.7 summarizes the prediction performance for six simultaneous trajectories, and Table 5.8 for nine. In places where the trajectories are close to each other, interference could result in data loss. At a node that is currently sampling one of the targets, data from one target that is

Table 5.7: Performance for trajectories 1, 2, 3, 4, 5, and 6

TP	**Path**	Data Loss (195 events)					Average Delay				
	Conf.	**Static**	**A1**	**A2**	**A3**	**A4**	**Static**	**A1**	**A2**	**A3**	**A4**
1	2	3	4	5	6	7	8	9	10	11	12
1	1	89					920				
	2	83	29	41	30	18	940	363	435	509	350
	3	45	(16)	(4)	(15)	(27)	407	(11%)	(-7%)	(-25%)	(14%)
2	1	109					876				
	2	83	37	46	35	24	834	418	377	436	350
	3	53	(16)	(7)	(18)	(29)	369	(-13%)	(-2%)	(-18%)	(5%)
3	1	96					945				
	2	86	20	30	20	9	869	395	427	472	434
	3	36	(16)	(6)	(16)	(27)	434	(9%)	(2%)	(-9%)	(0%)
4	1	84					915				
	2	91	40	55	38	26	991	481	492	539	450
	3	64	(24)	(9)	(26)	(38)	447	(-8%)	(-10%)	(-21%)	(0%)

Table 5.8: Performance for all trajectories

TP	**Path**	Data Loss (264 events)					Average Delay				
	Conf.	**Static**	**A1**	**A2**	**A3**	**A4**	**Static**	**A1**	**A2**	**A3**	**A4**
1	2	3	4	5	6	7	8	9	10	11	12
1	1	140					892				
	2	128	61	81	65	49	792	288	350	447	280
	3	88	(27)	(7)	(23)	(39)	286	(-1%)	(-22%)	(-56%)	(2%)
2	1	153					844				
	2	131	63	75	62	48	857	460	364	508	364
	3	83	(20)	(8)	(21)	(35)	316	(-46%)	(-16%)	(-61%)	(-16%)
3	1	137					863				
	2	133	44	67	46	37	718	297	318	384	290
	3	74	(30)	(7)	(28)	(37)	286	(-4%)	(-11%)	(-34%)	(-1%)
4	1	133					910				
	2	138	62	85	69	51	949	531	521	539	435
	3	94					469	(-13%)	(-11%)	(-15%)	(-7%)

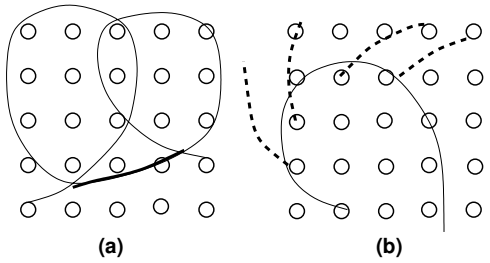

Figure 5.10: Data loss at Trajectory 6

Table 5.9: Performance for Trajectory A

TP	**Path**	Data Loss (171 events)					Average Delay				
	Conf.	**Static**	**A1**	**A2**	**A3**	**A4**	**Static**	**A1**	**A2**	**A3**	**A4**
1	2	3	4	5	6	7	8	9	10	11	12
1	1	49					864				
	2	47	20	20	14	2	942	510	383	414	383
	3	20	(0)	(0)	(6)	(18)	383	(-33%)	(0%)	(-8%)	(0%)
2	1	35					771				
	2	50	24	22	12	6	970	436	351	349	299
	3	22	(-2)	(0)	(10)	(16)	345	(-26%)	(-2%)	(-1%)	(13%)
3	1	47					876				
	2	41	30	32	10	6	789	537	403	425	356
	3	33	(3)	(1)	(22)	(27)	406	(-32%)	(1%)	(-5%)	(12%)
4	1	61					955				
	2	38	24	29	12	12	901	555	435	469	436
	3	30	(6)	(1)	(18)	(18)	441	(-26%)	(1%)	(-6%)	(1%)

waiting in one of its buffers can be overwritten by a data packet from another target.

Long trajectories. Two trajectories, namely trajectories A and B were derived from trajectories four and six. These trajectories were made longer than the trajectories defined in Figure 5.7. Experimental analysis in this section includes comparisons of algorithms with respect to one another.

Trajectory A. Refer to Table 5.9 for the performance of this trajectory. The algorithms produce poor improvements in average delay for TPs one and four. Methods A2 and A4 produce no improvement for TP one and 1% for TP four. Algorithms A1 and A3 produce worse results for all TPs. Methods A3 and A4 give good improvements in data loss.

It is observed that algorithm A4 performs best with respect to all other algorithms in this trajectory for all four target points. It gives an improvement in data loss ranging from 60% to 90% with respect to the best case path configuration and the

Table 5.10: Performance for Trajectory B

TP	Path	Data Loss (204 events)					Average Delay				
	Conf.	Static	A1	A2	A3	A4	Static	A1	A2	A3	A4
1	2	3	4	5	6	7	8	9	10	11	12
1	1	58					865				
	2	59	31	36	21	11	989	479	448	436	428
	3	36	(2)	(0)	(6)	(7)	445	(-7%)	(-1%)	(2%)	(4%)
2	1	30					757				
	2	61	5	14	3	2	916	406	370	371	284
	3	14	(-2)	(0)	(4)	(6)	370	(-9%)	(0%)	(0%)	(23%)
3	1	44					880				
	2	54	23	29	16	7	846	476	423	450	404
	3	29	(1)	(0)	(5)	(6)	423	(-13%)	(0%)	(-6%)	(5%)
4	1	53					761				
	2	57	12	18	8	0	989	396	363	427	350
	3	18	(3)	(1)	(10)	(12)	373	(-6%)	(3%)	(-14%)	(7%)

improvement in average delay ranges from 0% to 13%. Compared to algorithm A1, A4 gives an improvement ranging from 50% to 90% in data loss and 21% to 34% in average delay. Compared to algorithm A2, A4 gives an improvement ranging from 59% to 93% in data loss and 0% to 15% in average delay. Compared to algorithm A3, A4 gives an improvement ranging from 0% to 86% in data loss and 7% to 16% in average delay.

Trajectory B. Refer to Table 5.10 for the performance of this trajectory. Even for this trajectory, Algorithms A3 and A4 give good improvements in data loss. Algorithm A1 produces a negative improvement in average delay for all TPs. Excluding algorithm A4 for target point two, no algorithm produces significant improvements in average delay for any target point.

Even in this case, algorithm A4 performs best with respect to all other algorithms for all four target points. It gives an improvement in data loss ranging from 69% to 100% with respect to the best case path configuration and the improvement in average delay ranges from 4% to 23%. Compared to algorithm A1, A4 gives an improvement ranging from 60% to 100% in data loss and 11% to 30% in average delay. Compared to algorithm A2, A4 gives an improvement ranging from 69% to 100% in data loss and 5% to 23% in average delay. Compared to algorithm A3, A4 gives an improvement ranging from 33% to 100% in data loss and 2% to 23% in average delay.

5.6.2 Performance of the four prediction algorithms

Since the quasi-static algorithm uses Bayesian inference to map to static trajectories, it performs best when the rate of change of velocity and angle is less than a threshold value, the condition that makes the trajectory assume a linear motion. Nodes are able to map to a static trajectory segment that closely resembles the actual trajectory. In other words, this algorithm performs well when previous readings can be used well to predict the future trajectory. This behavior is expected considering the properties of Bayes Theorem. Experiments show that this algorithm is very sensitive to small mispredictions when it comes to improving data loss, but improves performance significantly if the prediction is accurate. It was also observed that the algorithm improves average delay better than the trajectory bounded and stochastically bounded algorithms with a few exceptions. This is because the segment this algorithm maps to is a fairly accurate estimation of the direction of the trajectory flow even if it mispredicts the future sensing nodes.

The bounded trajectory and stochastically bounded trajectory methods define Trajectory Approximating Regions (TAR) to estimate the area in which the trajectory would lie in the future. The paths are selected based on the TARs and not on the estimated direction of trajectory motion. The algorithms improve average delay if a major portion of their TARs are occupied by the future trajectory. The stochastically bounded algorithm performs better than the quasi-static and bounded trajectory when the trajectory abruptly switches state (e.g. looping trajectories when $\delta\theta$ is very high) since it uses Markov Decision Processes and is fully independent of history. For these conditions, the bounded trajectory technique performs better than the quasi-static trajectory based technique, but not as well as the stochastically bounded trajectory technique.

For long trajectories, it is seen that the adaptive method produces results significantly better than the quasi-static algorithm, which makes Bayesian inference to make trajectory predictions similar to sensor selection techniques in [17]. Even in the general case, the adaptive method produces better results compared to the other algorithms for all trajectories since it was intuitively designed based on the experimental analysis of the first three algorithms. Hence, this technique is able to capture all the advantages of the other three techniques.

5.6.3 Experimental analysis of the adaptation policy design

Experiments on trajectories A and B were conducted by using the parameter optimization techniques discussed in the previous section. The focus was to improve data loss with a slight trade off on average delay by switching between different design points involving parameters namely buffer size, baud rate, and radio power. These experiments use the following design points (DPs):

1. Buffer size (4 DPs): $B_f = 1$ to 4 packets. The default value is 1.
2. Baud rate between Embedded Node and the Radio Module using SPI (3 DPs): B_r = 4.8kbps, 9.6kpbs and 19.2kbps. The default value is 9.6kbps.
3. Radio baud rate (2 DPs): 31.25kbps, 62.5kbps. The default value is 31.25kbps.
4. Radio power levels (8 DPs): $RP_r = 0$ to 7, 7 being the highest power level. The default value is 1.

We conducted four sets of experiments on parameter optimization using buffer size selection. The experiments used fixed path configurations and the trajectory predicted by the adaptive algorithm. Each set imposed a constraint on the maximum value of buffer size. A buffer constraint of one, as expected, did not improve data loss and average delay. A buffer constraint of two improves data loss by 32% to 35% with an increase in average delay by 10%. A buffer constraint of three improves data loss by 55% with an increase in average delay by 16%. A buffer constraint of four improves data loss by 65% to 70% with an increase in average delay by 20%.

Three sets of experiments were performed on baud rate optimization with no constraints on the utility of baud rate. The three sets of experiments correspond to three different radio power levels. These experiments are synthetic experiments where processing time was adjusted to a value that is in the same order of magnitude as that of the baud rate values. Baud rate optimization techniques improved data loss by 15% to 50%. The reason for this wide range is due to the variable speed of the trajectory. It was observed that baud rate optimization is more effective for faster trajectory speeds. The increase in average delay with respect to the default baud rate value is 15% to 35%. It was observed that changing radio power results in the same improvement with respect to the default values for that radio power level. Switching to a different power level does not change data loss, but improves average delay due

to less number of retransmissions. With respect to power level 1, power levels 4 and 7 improve average delay by 40% and 45% respectively.

5.7 Conclusions

This chapter describes adaptation policy design that optimally selects network parameters that include data paths and network resources that include memory buffer size and baud rates to satisfy the performance requirements of the application. Unlike Chapter 4 which the purpose was tracking quasi-static entities, this chapter deals with dynamic entities where the distributed variables move within the space with certain properties, specifically velocity v and direction θ. Depending on the correlations between data paths and trajectory of the moving entity, data paths and other network parameters are switched to optimize performance by reducing latency and data loss for building highly accurate data models. Depending on the nature of the trajectory which relates to the rate of change of velocity δv and direction $\delta\theta$, four trajectory prediction algorithms were devised that predicted the future trajectory and each of these algorithms perform better than the others under certain conditions.

The four algorithms are: (i) quasi-static prediction based on Bayesian inference, (ii) bounded trajectory method which approximates trajectories as sequences of bounded convex - concave regions, (iii) stochastically bounded trajectories, which describes the transitions from a bounded region to another follows as a Markov process, and (iv) an adaptive model, which perform a dynamic linearization of trajectories. The accuracy of the predictions depends on the assumptions made by each of the algorithms and the dynamic nature of the entity. The performance of each algorithm under different conditions were derived based on experimental analysis.

Bayesian-based trajectory prediction performs well for quasi-static trajectories but gives lesser improvements for long, complex trajectories and provides better improvements in latency compared to the bounded and stochastically bounded trajectory techniques, these result in smaller data loss compared to Bayesian-based trajectory predictions. The adaptive prediction method reduces data delay between 12% and 36%, and data loss up to 90% for long trajectories. Increased buffer sizes reduce data loss up to 70% but also increase delay by 20%. Baud rate optimization improves data loss between 15% to 50% and is more effective for fast trajectories. Increased radio power levels reduce average delay by up to 45%.

Chapter 6

Distributed Data Modeling and Model Parameter Lumping for CPS Applications

6.1 Introduction

Many modern applications require dependable actuation and control based on physical data collected from distributed areas [56]. Such applications include gas distribution in power plant control [92], sound in vehicle tracking [26], temperature monitoring in warehouses [93], smart oil fields [50], distributed gas monitoring [101], but also future generation transportation systems, intelligent power grid, health care, homeland security, and many more [56, 26]. Data collection is through a network of embedded sensors that acquire with different resolutions physical data, like temperature, humidity, gas composition, object proximity, light intensity, and magnetic field. The embedded sensing nodes are connected through wired and/or wireless networks. Nodes also interact sometimes through shared physical media, such as the air mass in a room, the local electromagnetic field, or the human body [26].

Such applications require accurate data representations and models based on data collected over space and time. The representation and models have to be built in real-time starting from raw data and then used for optimized decision making. Data models are associated to curves, 2D areas, and 3D volumes, and not only to singular points as in traditional embedded systems. These data models generate model pa-

rameters that would describe the manner in which data is distributed in the region, establish data correlations and relations that pertain to physical laws of the monitored entities. The energy at each node in the network is represented as a state variable and each state variable is expressed in the form of differential equations, discussed in the later sections, that contain these model parameters. The generated model parameters correspond to *gradient coefficients*, that model flow of data between nodes in the network, and *storage coefficients* that model the energy stored at a node.

Apart from producing accurate data models starting from raw sampled data, there is also the need to tackle the performance and resource constraints specified by the high-level specification notations discussed in Chapter 2. These constraints provide limited network resources and even if the procedure is able to generate accurate data models locally at each state variable, these model parameters have to reach the decision making node within the timing constraints for precise data representations. Traditional communication topologies, like minimum height trees, can increase the data volume at nodes, thus result in high data losses if the local memory resources are exceeded. The adaptation policies discussed in Chapter 4 and Chapter 5 are used to select the right resources that should be utilized to transmit these model parameters to the target point. In addition, this chapter also discusses Data lumping [81], the process of producing cumulative representations of locally produced data models while they are transmitted to the target point. However, while lumping the model parameters along the data paths, careful decisions have to be made to avoid loss of information at the critical points when these model parameters are lumped. In addition, important data correlations can be lost depending on information gradients within the network and the utilization rates of the communication paths. All these factors have to be taken into consideration during the process of modeling distributed sampled data for efficient decision making in dynamic conditions. This chapter focuses on building these model parameters and characterizing the lumping and correlation errors. A method for optimized distributed data lumping is also discussed where the cost function provides importance to either optimizing lumping errors or latency. Experiments were run on a case study that generated temperature values using a simulator [114] for the floorplan given in Figure 6.3 that corresponds to the SUN Niagara T1 architecture [113]. The experiments section generated four datasets that cover different dynamics of entities (heat sources for the case study) and characterizes modeling errors after thoroughly analyzing the effects of the dynamics of entities on

these modeling errors. Experiments were also run on three datasets, one with static heat sources, the second one with moving heat sources, and the third one with heat sources randomly appearing and leaving a core.

6.2 Related Work

The process of modeling has been extensively used in thermal modeling of heat generated in ICs. Sridhar et al. [105] present a model for thermal simulation of 3D ICs with multiple inter-tier microchannel liquid cooling and try to accurately predict the temperatures at different points in a 3D chip using the thermal properties (thermal conductivity and capacitance) of Silicon and other materials used at different elements in the architecture. The 3D-chip was divided into thermal cells and heat transfer was modeled according to the material properties and the heat generated at those cells based on input power values and the rate of pumping liquids across microchannels. Hence, the purpose of modeling in existing techniques is thermal simulation to generate temperature values over time [105, 112] at different points in the chip to aid thermal management [106]. Thermal management through voltage and frequency scaling and task scheduling has been extensively studied [108, 109, 111], but these management policies are centralized. On the other hand, the purpose of modeling in our application is using sensor data distributed in an area to extract model parameters that would represent the phenomenon.

The process of distributed data lumping presented in this work differs in several ways from the aggregation techniques proposed for wireless sensor networks (WSNs). The main goal of cumulative data representation for WSN is to improve the network performance, e.g., bandwidth, throughput, and energy consumption [95, 96, 97, 99]. A feedback-based scheme is discussed in [95]. The scheme adapts to changing traffic conditions and time requirements. Tiny Aggregation (TAG) offers services for distributing and executing aggregation queries over WSNs [100]. Parents compute monotonic aggregates based on data sent by their children over a tree network. The work in [91] reduces data communication by fitting functions that approximate the transmitted values. SPIN proposes a procedure to avoid transmission of redundant data [96]. Redundancy elimination through coding is discussed in [102]. Recent work in [104] defines a centralized threshold-OR fusing rule for combining sensor samples under normally distributed, independent additive noise conditions. Xue et al. [103]

propose a locally weighted fusion function for improving model accuracy. Fuzzy set based fusion of classifier outputs is discussed in [94]. In contrast, control applications require representations that are produced based on application-specific functions expressing the physical environment rather than on generic functions [98]. Models are organized at multiple levels of abstractions characterized by different representation accuracies. Moreover, optimization methods are needed to tackle the performance requirements and resource constraints of the application.

6.3 Distributed Data Modeling for CPS

Physical models are built by discretizing the space, introducing state variables for every node of the discrete mesh, and expressing the conservation and rate of change laws for the state variables [13]. Every node in the grid is described by continuous parameters that are of two kinds: (a) *Across parameters* that model flow between physical points, and (b) *Through parameters* that describe the derivative and integration operations.

The differential equations are derived starting from the law of energy conservation given by equation 6.1:

$$\frac{E_i(t) - E_i(t-\delta)}{\delta} = \sum_k \dot{E}_{i,k}(t)) + \sum_k \dot{E}_{k,i}(t)) + \dot{E}_{in\_out,i}(t) \tag{6.1}$$

The left hand side of the equation indicates the rate of change of energy at a node i. The first two terms on the right hand side indicates the rate of transfer of energy between the node i and the neighbors k. The term $\dot{E}_{i,k}(t)$ indicates the rate of transfer of energy from node i to node k, $\dot{E}_{k,i}(t)$ indicates the rate of transfer of energy from node k to node i, and $\dot{E}_{in\_out,i}(t)$ indicates the rate of adding (or removing) energy to (from) node i.

The terms $\dot{E}_{i,k}(t)$ and $\dot{E}_{k,i}(t)$ indicate energy flux ($f_{i,k}$ and $f_{k,i}$), and this rate of energy transfer (flux) depends on the difference of physical data sensed at nodes i and k (given by $Y_i(t)$ and $Y_k(t)$), and the transfer coefficient (given by $G_{r_{i,k}}$ and $G_{r_{k,i}}$). We call these transfer coefficients as *gradient coefficients*. The difference in energy $E_i(t) - E_i(t-\delta)$ over time δ causes an increase or decrease in the value of the physical data (the change to value $Y_i(t)$ from $Y_i(t-\delta)$) and the amount of this change depends on the capacity of the node to store data (given by S_{c_i}). We call this S_{c_i} coefficient as

storage coefficient. Hence, equation 6.1 can be replaced by the differential equation given by equation 6.2.

$$S_{c_i} \frac{[Y_i(t) - Y_i(t-\delta)]}{\delta} = \sum_k [\frac{Y_k(t) - Y_i(t)}{G_{r_{k,i}}}] + \sum_k [\frac{Y_i(t) - Y_k(t)}{G_{r_{i,k}}}] + [\dot{E}_{in,out,i}] \tag{6.2}$$

where k indicates the neighbors of the node and δ is a very short interval of time, Gradients $G_{r_{i,k}}$ and $G_{r_{k,i}}$ represent gradient coefficients that model flow of data between nodes k and i. Storage coefficient S_{c_i} represents the capacity of a node to store energy and indicates the rate of increase or decrease of Y_i over time δ for a difference in energy $E_i(t) - E_i(t-\delta)$. $\dot{E}_{in,out,i}(t)$ is the rate of change of energy incident at node i due to adding or removing energy from the environment modeled by the input source. The gradient coefficients Gradients $G_{r_{i,k}}$ and $G_{r_{k,i}}$ are combined to form an equivalent gradient coefficient value $G_{req_{i,k}}$ for a node i with respect to neighbor k and equation 6.2 is simplified to produce equation 6.3.

$$S_{c_i} \frac{[Y_i(t) - Y_i(t-\delta)]}{\delta} = \sum_k [\frac{Y_k(t) - Y_i(t)}{G_{req_{i,k}}}] + [\dot{E}_{in,out,i}] \tag{6.3}$$

Starting from raw data samples that correspond to the Y_i and Y_k values, the differential equation 6.3 is used to compute the unknowns that correspond to the model parameters S_c and G_r values over time. The values of these parameters change depending on how dynamics and other properties of the physical entities that cause the change in measurements. Hence, by solving the above differential equation for all nodes in the network using sensor data measured over time, the dynamics of all the state variables over space and time can be predicted, which can be used to devise dynamic decision making strategies. The distributed model produced from equation 6.3 is shown in Figure 6.2(a), where the gradient coefficients are considered analogous to resistances and the storage coefficients are analogous to capacitances.

Since data in CPS is distributed in nature, data is transmitted along data paths in the form of tuples. Each tuple contains information regarding node position, time of sampling, the attributes of the sampled signal and the computed model parameters. Thus, each node communicates data with its own neighbors, locally produces the model parameters that include the through and across variables and then transmits

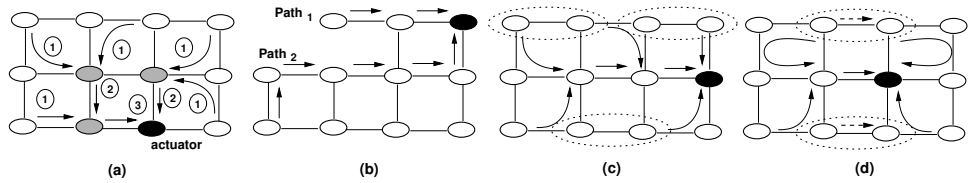

Figure 6.1: Distributed data collection and lumping

this information in the form of tuples to the target point.

While transmitting this data along the path, the process of lumping is performed at different abstraction levels (granularities) and it introduces errors that depends on various factors. Higher abstraction levels introduce higher errors (as more information is lost when intermediate nodes are removed), but they decrease the data communication volume. This reduces the utilized resources (such as communication bandwidth, power, memory, etc.) and helps meeting the timing and resource constraints of the application.

Optimizing distributed data lumping is challenging and the procedure must decide among multiple data paths available for collecting data. For example, tree networks, as in Figure 6.1(a), have minimum height, which reduces communication time. But such networks have an increased data volume at certain nodes (the gray nodes). To reduce data loss, the gray nodes must have larger input and output buffers, and support higher bandwidth rates on their outgoing links. The height of critical paths can be reduced by allowing more incoming paths to converge at a node. This comes, however, at the penalty of more complex interfacing for the nodes.

The data paths influence the modeling and lumping errors as they decide what data is available at each node. In Figure 6.1(b), the overall error over time ΔT is a function of the errors e_1 and e_2 for each of the two links: $error_{total} = \int_{\Delta T} f(e_1, e_2)dt$. $error_{total} \leq precision_{total}$, the overall precision requirement. In addition, the errors along each path might be less than given threshold values: $\int_{path_1} g(e_1)dt \leq P_{th1}$, and $\int_{path_2} g(e_2)dt \leq P_{th2}$. The total additive error is: $error_{total} = \sum_{\Delta T} \frac{\sum_{path_1} e_i + \sum_{path_2} e_j}{N}$. Error along path1 is $\sum_{path_1} \frac{e_i}{N_1} \leq P_{th1}$ and for path2, it is given by $\sum_{path_2} \frac{e_j}{N_2} \leq P_{th2}$. $N_{1(2)}$ is the number of sampling nodes along a path, and $N = N_1 + N_2$. There are many ways of selecting the acceptable local errors $e_{i,(j)}$ at the sensing nodes while meeting all constraints, however each distribution has different timing performance. For example, larger local errors at the early node allow more aggressive lumping, thus less data needs to be propagated along the paths but the trade-off is that the early

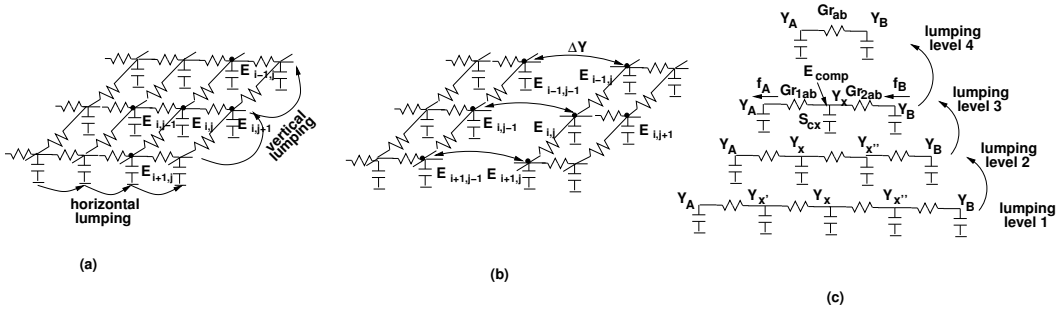

Figure 6.2: Correlated attributes: (a) modeling, (b) path-induced errors and (c) lumping errors

nodes process more and less error margin is available to the latter nodes.

In Figure 6.1(c), data is collected and transmitted along the shown paths. As a result, certain data correlations are missed, such as those between data collected at the node pairs marked with a dotted line. The amount of missed correlations can be reduced by selecting the data path in Figure 6.1(d).

The above mentioned errors that are produced due to loss of information during lumping, the missed data correlations due to the nature of the data paths, and the sensing errors due to sensor limitations and time synchronizations are characterized in the next section.

6.4 Modeling Errors

Physical data is modeled locally at each node using differential equations formulated at equation 6.3 [13]. The nature of the data correlations and the utilization rates of the communication paths influence modeling and lumping errors.

This section models describes the different types of modeling errors that are generated depending on the procedure of lumping, the correlation between data paths and the dynamics of the physical entity, and the limitations at the sensing front end. The errors are represented in the form of equations and these equations are used to derive the error bounds that characterize these errors.

6.4.1 Lumping Errors

All model parameters cannot be streamed to the target point within the given timing constraints of the application and the limited availability of the network resources,

specifically communication bandwidth and input memory buffers. This establishes the need for lumping model parameters along the path and the locally produced model parameters are lumped to produced a more global model. This process of lumping results in errors due to loss of information when intermediate state variables are removed. The error depends on the level of abstraction used during the process of lumping.

Figure 6.2(c) shows the distributed parameter description used to express the lumping error. The description is a tree network where every tree corresponds to a data collection path. Lumping simplifies the cascaded stages by eliminating state variables. The introduced error is estimated by comparing the behavior of the initial and lumped network, such as the two top networks in Figure 6.2(c). After solving the equations for the top two networks in Figure 6.2(c), the following relationships result for energy flux f and sample values Y:

$$Error\ f_A(t) = \gamma\ S_{c_x}\ \frac{\gamma\ Y_A(t) + Y_B(t) - (\gamma+1)\ Y_x(t-\delta) - (\gamma+1)\ \delta\ (E_{comp}(t)/S_{c_x})}{(\gamma+1)^2\ \delta + \gamma\ S_{c_x}\ G_{r_{ab}}} \quad (6.4)$$

$$Error\ f_B(t) = -S_{c_x}\ \frac{\gamma\ Y_A(t) + Y_B(t) - (\gamma+1)\ Y_x(t-\delta) - (\gamma+1)\ \delta\ (E_{comp}(t)/S_{c_x})}{(\gamma+1)^2\ \delta + \gamma\ S_{c_x}\ G_{r_{ab}}} \quad (6.5)$$

where $E_{comp}(t)$ is the summation of energy due to the missing links and external input energy. The expression for lumping error at a node (state variable), $Err_{x,lump} = \Delta Y_x(t)$ is derived by subtracting the expression for $\mathtt{Y_x(t)}$ before lumping from the expression for $Y_x(t)$ after lumping.

$$Err_{x,lump} = \kappa\ (\gamma\ Y_A(t) + Y_B(t) - (\gamma+1)\ Y_x(t-\delta) - (\gamma+1)\ \delta\ (E_{comp}(t)/S_{c_x})) \quad (6.6)$$

$$\kappa = \frac{\gamma\ S_{c_x}\ G_{r1_{ab}}}{(\gamma+1)^2\ \delta + \gamma\ S_{c_x} G_{r_{ab}}} \quad (6.7)$$

Gradient coefficient $G_{r2_{ab}}$ is assumed to be equal to γ times the value of $G_{r1_{ab}}$. δ is the time discretization step used in distributed parameter modeling. Note that ΔY_x characterizes the error of the removed state variable, and Δf_A and Δf_B the errors of the energy flow along the two directions along the select path. $Y_x(t-\delta)$ is the sensor reading at the previous time moment. For the lumped network, Y_x is approximated as $Y_x = \frac{\gamma\ Y_A + Y_B}{\gamma+1}$. For situations where $G_{r1_{ab}}$ and $G_{r2_{ab}}$ are equal, the value of γ is 1.

Lemma: The equation that bounds the maximum increase in lumping errors for lumping state variable X over W samples is derived from equation 6.6

$$bound_{W,lump} \leq \kappa\ W\ \delta\ (bound_{lump1} - bound_{lump2}) \tag{6.8}$$

where,

$$bound_{lump1} = (\gamma+1)\ \delta\ \ddot{Y}_{x,MAX} + \gamma\ (\dot{Y}_{A-x_{MAX}}) + (\dot{Y}_{B-x_{MAX}}) \tag{6.9}$$

and

$$bound_{lump2} = \frac{(\gamma+1)\ \delta}{S_{c_x}}\ \dot{E}_{comp_{x,MIN}}) \tag{6.10}$$

where $\ddot{Y}_{x,MAX}$ is the maximum of the second derivative of the data at node x, $\dot{Y}_{A-x_{MAX}}$ and $\dot{Y}_{B-x_{MAX}}$ are the maximum first order derivative of the difference in sensor readings of node x with respect to neighboring nodes A and B in the path. Hence, $Y_{A-x}(t) = Y_A(t) - Y_x(t)$, and $Y_{B-x}(t) = Y_B(t) - Y_x(t)$, and $bound_{W,lump} = error_{Y_x}(t) - error_{Y_x}(t - W\ \delta)$.

The total error depends on the number of removed state variables in the path and the variation characteristics of those variables. For slowly changing variables, more variables can be removed while keeping the error below a limit. For different variation characteristics, the maximum lumping level can be found to stay within a certain error. The lumping errors are characterized at the end of the subsection *Analysis of Bounds for Lumping Errors* in the experiments section. The paragraph under the heading *Characterizing Lumping Errors* provides conclusions from the analysis of experimental results that were obtained from different datasets which had entities stationary and moving within the network. The conditions when lumping errors might

increase or decrease significantly are mentioned.

6.4.2 Correlation errors

Correlation errors are introduced because not all tuples are calculated due to the "blind" spots introduced by the data collection paths. Figure 6.1(b) explains how missing correlations occur between neighboring sensing nodes. The absent tuples correspond to data from neighboring nodes that belong to different collection paths. Hence, correlation errors are path-induced errors and are modeled as follows:

$$Err_{b,path} = \sum_{\forall b} \sum_{\forall p}^{NL} \sum_{c \in p} error_{(p)} \tag{6.11}$$

b is a pair of neighboring nodes i and k that belong to different paths. $error_{(p)}$ is the error introduced due to the missing tuples c at all lumping levels p of node k (assuming that data is transmitted from node i to node k).

The error introduced by a collection path is estimated as shown in Figure 6.2(b). A path eliminates the coupling gradient coefficients between two columns, such as the gradient coefficients between storage variables $S_{c_{i-1,j-1}}$ and $S_{c_{i-1,j}}$, between storage variables $S_{c_{i,j-1}}$ and $S_{c_{i,j}}$, and so on. This removes the correlation between the associated state variables, such as energy values $E_{i-1,j-1}$ and $E_{i-1,j}$, between $E_{i,j-1}$ and $E_{i,j}$, etc. The error associated to decoupling columns $j-1$ and j is proportional to the overall impact of the flux through the removed gradient variables:

$$Error_{j-1,j,path} = \sum_{i} [\frac{E_{i,j}/S_{c_{i,j}} - E_{i,j-1}/S_{c_{i,j-1}}}{G_{r_{i,(j-1,j)}}}] \tag{6.12}$$

$$E_{i,j}/S_{c_{i,j}} = G^{eq}_{r_{i,j}} f^{eq}_{i,j} + E^{eq}_{i,j}/S^{eq}_{c_{i,j}} \tag{6.13}$$

$$E_{i,j-1}/S_{c_{i,j-1}} = G^{eq}_{r_{i,j-1}} f^{eq}_{i,j-1} + E^{eq}_{i,j-1}/S^{eq}_{c_{i,j-1}} \tag{6.14}$$

Parameters $G^{eq}_{r_{i,j-1}}$ and $S^{eq}_{c_{i,j-1}}$ are the equivalent coefficient values of the left subnetwork (after decoupling) measured from a decoupled node to the actuator. Simi-

larly, parameters $G^{eq}_{r_{i,j}}$ and $S^{eq}_{c_{i,j}}$ are the equivalent coefficient values of the right sub-network. $G_{r_{i,(j-1,j)}}$ are the removed gradient coefficients due to the missed correlations induced by the collection paths.

Referring to the network at lumping level 3 in figure 6.2(c), the expression for correlation errors is derived and is given by:

$$Err_{x,correlate} = \frac{1}{\gamma\ G_{r1_{ab}}}\ [expr_{x,correlate1} - expr_{x,correlate2}] \tag{6.15}$$

$$expr_{x,correlate1} = \gamma\ S_{c_x}\ G_{r1_{ab}}\ \dot{Y}_x(t) + (\gamma + 1)\ Y_x(t) \tag{6.16}$$

$$expr_{x,correlate2} = \gamma\ Y_A(t) + Y_B(t) + \gamma\ G_{r1_{ab}}\ E_{in,out}(t) \tag{6.17}$$

where $E_{in,out}(t)$ is the external input energy source incident at the node.

Lemma: The correlation error at state variable Y_x over W samples under the assumption that the same path configuration is used is bounded by the following expression where $bound_{W,correlate} = Err_{x,correlate}(t) - Err_{x,correlate}(t - W\ \delta)$:

$$bound_{W,correlate} \leq \frac{W\ \delta}{\gamma\ G_{r1_{ab}}}\ [bound_{W,correlate1} - bound_{W,correlate2}] \tag{6.18}$$

where,

$$bound_{W,correlate1} = \gamma\ S_{c_x}\ G_{r1_{ab}}\ \ddot{Y}_{x,MAX} \tag{6.19}$$

and

$$bound_{W,correlate2} = \gamma\ \dot{Y}_{A-x_{MIN}} + \dot{Y}_{B-x_{MIN}} + \gamma\ G_{r1_{ab}}\ \dot{E}_{in,out,MIN} \tag{6.20}$$

The correlation errors are characterized at the end of the subsection *Analysis of Bounds for Correlation Errors* in the experiments section. The paragraph under the heading *Characterizing Correlation Errors* provides conclusions from the analysis of

experimental results that were obtained from different datasets which had entities stationary and moving within the network. The conditions when correlation errors might increase or decrease significantly are mentioned.

6.4.3 Collection Errors

Collection errors are introduced if the sensor data cannot be acquired due to the hardware constraints of the sensing frontends. For example, not all sampled values present in the input buffers can be processed. Lets assume that error Err_{sens_j} is due to discarding one physical value of sensor $sens_j$. The total error introduced by all discarded data at the node's sensors is:

$$Err_{j,basic} = \sum_{\forall sens_j} \lambda_j \; q_{sens_j} \; Err_{sens_j} \tag{6.21}$$

and

$$Err_{j,basic} = \sum_{\forall sens_j} \lambda_j \; q_{sens_j} (\sum_{\forall k} error_{(k)}) \tag{6.22}$$

q_{sens_j} is the sampling rate of the sensor. λ_j is the rate of discarding values at sensor $sens_j$ ($\lambda_j < \alpha_{sens_j}$). $error_{(k)}$ is the error introduced for tuple k by the discarded data.

Another type of collection error corresponds to time synchronization errors that are introduced due to dissimilar clocks. Even if the nodes have the same sampling rate, they produce data at different time instances since their clocks are not synchronized. Considering the network at lumping level3 in figure 6.2(c), models are produced and updated by computing model coefficients from data produced at all three nodes at a particular time instance t, i.e. the G_r and S_c values are computed using the values of $Y_A(t)$, $Y_B(t)$ and $Y_x(t)$. Assuming that available sensor values are $Y_A(t)$, $Y_B(t)$ and $Y_x(t - dt)$, where the sensor reading available at node x is time shifted by a factor of dt due to unsynchronized clocks, we model the introduced errors by comparing the behavior of the synchronized network with that of the unsychronized network. δ is the time discretization step used in distributed modeling.

$$Err_{x,sync} = \Delta f_x(t) = 2 \; S_{c_x} \; \frac{Y_x(t) - Y_x(t - dt)}{S_{c_x} G_{r1_{ab}}} \tag{6.23}$$

where,

$$Y_x(t) - Y_x(t - dt) = dt \ \frac{\gamma Y_A(t) + Y_B(t) - (\gamma + 1)\ Y_x(t - dt) + \gamma\ (E_{comp}(t)\ G_{r1_{ab}})}{dt\ (\gamma + 1) + \gamma\ S_{c_x}\ G_{r1_{ab}}} \quad (6.24)$$

It is observed that time synchronization errors produced due to a smaller factor dt is negligible compared to lumping and correlation errors.

6.5 Constraint Modeling

The purpose of model parameter lumping discussed in the previous sections is cumulative data representation to reduce communication overhead. During this process, intermediate nodes are lumped at different levels of hierarchy shown in Figure 6.2(c). The tradeoff experienced during the process of lumping is the loss of information contained in those nodes resulting in lumping errors. This section formulates the performance and design constraints for distributed data lumping for obtaining the right resource parameters and lumping hierarchy depending on the nature of the cost function. The resource parameters include the rate of using different bandwidth values and the lumping hierarchy value corresponds to the probability that a node would be lumped given an error value.

The equations for performance and design constraints given below are specific for a data path (DAP) with different path segments ($p_x \in DAP$).

6.5.1 Sensing and input network interfacing

The average input data rate from all sensors of a node x is given by equation 6.25.

$$Input\_rate_x = \sum_{\forall sens_j} \alpha_{sens_j}\ q_{sens_j} \quad (6.25)$$

$\alpha_{sens_j} \geq 0$ is the rate of using sensor j and the sensing rate is greater than the required resolution, $q_{sens_j} \geq Resolution_j$. NET_IN_x is the amount of data input q_x^{in} from other nodes in the path.

$$NET\_IN_x = q_x^{in} \tag{6.26}$$

6.5.2 Output network interfacing

The amount of data output from node x is NET_OUT_x and data loss occurs when the input rate is higher than the output rate and the difference is more than the available buffersize $buff$. Hence, the constraint equation 6.27 requires the buffer size to be larger than the difference to avoid data loss.

$$buff \geq (Input\_rate_x + NET\_IN_x) - NET\_OUT_x \tag{6.27}$$

Assuming the input rate is the same as output rate, the equation for NET_OUT_x depends on the amount of data output by the sensing module and the amount of data input from all previous nodes $prev_x$ in path segment p_x

$$NET\_OUT_x = prob_{lump_x}\ DATA\_OUT_x + \sum_{\forall prev_x \in p_x} prob_{lump_x}\ DATA\_OUT_{prev_x} \tag{6.28}$$

where the $DATA\_OUT_x$ is the size of a packet in bytes sent out by node x and $prob_{lump_x}$ is the probability that a node x would be lumped and is given by equation 6.29.

$$prob_{lump_x} = \sum_{\forall j} \lambda_j\ lump\_level_j \tag{6.29}$$

where $lump\_level_j$ corresponds to the lumping level, which is defined by the percentage of lumped nodes in p_x and λ_j corresponds to the utilization rates of using $lump\_level_j$ and $\sum_{\forall j} \lambda_j = 1$.

6.5.3 Path delay

The delay of path p_x is the sum of the average execution time of all nodes x along the path plus the average time for transmitting the output data of each node:

$$Delay_{p_x} = \sum_{\forall x \in p_x} (Delay_x + Delay_x^{out}) \tag{6.30}$$

The average execution time of node x is equal to:

$$Delay_x = \sum_{\forall j} \alpha_j \; Exec_j \tag{6.31}$$

$Exec_j$ is the execution time of primitive j executed by node x, and α_j is the execution rate of the primitive. Each primitive may correspond to clock frequency.

The average time for transmitting the output data NET_OUT_x of node x is expressed as follows:

$$Delay_x^{out} = NET\_OUT_x \; \sum_{\forall j} \beta_j \; \frac{1}{BW_j} \tag{6.32}$$

β_j is the rate of using bandwidth BW_j for the output link and $\sum_{\forall j} \beta_j = 1$.

$Latency_{aver}$ is the delay of the longest path segment p_x of a DAP.

$$Latency_{aver} = \max_{p_x} Delay_{p_x} \tag{6.33}$$

6.5.4 Total Lumping Error

The total lumping error is given by equation 6.34 below.

$$Err_{lump,total} = \sum_{\forall x \in DAP} prob_{lump_x} \; Err_{x,lump} \tag{6.34}$$

where the $Err_{x,lump}$ values are computed using equations 6.6 and 6.7 through profiling and $prob_{lump_x}$ is given by equation 6.29.

6.5.5 Cost Function

$$Cost = \zeta \; Err_{lump,total} + \mu \; Latency_{aver} + \eta \; Energy_{total} \tag{6.35}$$

Given a set of data paths (DAP), equations 6.25- 6.35 are solved to compute the λ_j and β_j values that correspond to utilization rates of different lumping levels and bandwidth values.

6.5.6 Optimizing lumping error

Based on the given cost function (equation 6.35), the probability of lumping a node $prob_{lump_x}$ (equation 6.29) is obtained by computing the λ_j values. A threshold value of lumping error $threshold_{lump}$ is computed based on this value of $prob_{lump_x}$, the typical value of lumping error obtained through profiling and the size of the path segment p_x. A high ζ to μ ratio in the cost function would produce a high threshold value since tolerance to high lumping level is less. A low ratio of ζ to μ corresponds to giving higher priority to improving latency, thus increasing the tolerance for lumping errors.

The decision making procedure that determines whether to lump a node considers more number of factors in addition to considering the lumping error computed using equations 6.6 and 6.7. Equations 6.36- 6.39 describe the procedure that makes the lumping decision by computing the value of $decision_{lump_x}$ and comparing its value to $threshold_{lump}$.

$$decision_{lump_x} = W_1 \; factor1_{lump} + W_2 \; factor2_{lump} + W_3 \; factor3_{lump} + W_4 \; Err_{x,lump} \tag{6.36}$$

where,

$$factor1_{lump} = 1 - \frac{\ddot{Y}_{x,MAX} - \ddot{Y}_x}{\ddot{Y}_{x,MAX}} \tag{6.37}$$

$$factor2_{lump} = 1 - \frac{\dot{Y}_{A-x_{MAX}} - \dot{Y}_{A-x}}{\dot{Y}_{A-x_{MAX}}} \tag{6.38}$$

$$factor3_{lump} = 1 - \frac{\dot{Y}_{B-x_{MAX}} - \dot{Y}_{B-x}}{\dot{Y}_{B-x_{MAX}}} \tag{6.39}$$

$Err_{x,lump}$ is computed using equations 6.6 and 6.7. The equation for the first factor, equation 6.37, tries to capture the entry of an entity that might result in very high value of double derivative of sensor reading. This value remains high until it reaches steady state. The second and the third factors, equations 6.38 and 6.39, correspond to the difference of sensor readings of the node with respect to the previous and next nodes in the path. Lumping the node would result in losing information with respect to gradients between the nodes. Areas where gradient values are high might be helpful in decision making procedures and hence, the node should not be lumped in case of high gradient values. The contribution of these factors become very important when an entity enters the vicinity of one of its neighbors. Even though the lumping error $Err_{x,lump}$ is typically low under these circumstances, lumping this node would affect the gradient coefficients of its neighbors and result in important loss of information at the neighboring node. The weights W_1, W_2, W_3 and W_4 are computed based on current conditions, although more emphasis is given on the fourth factor. The result is used along with the value of $threshold_{lump}$ to decide whether to lump a node.

6.6 Experiments

Experiments were run on 4 datasets created on the floorplan given in figure 6.3. The floorplan has 8 cores, 4 L2 caches (1 memory cache shared by 2 cores), 2 memory elements, one clock module, one floating point unit, crossbar interconnect and other processing elements. The architecture is the ULTRASPARC Niagara T1 architecture [113]. Datasets were generated using a simulator [114].

The 25-node network contains one sensor located at each processing element, except cache0 and cache2 which have 2 sensors. 4-D plots are given where data and modeling errors at different time instances are plotted over the network region. 4-D plots are given in this chapter and Appendix B and these plots are contour plots that represent either temperature values or percentage modeling errors at the sensor nodes spread across the network. The x-axis of the plots represents the x-coordinates

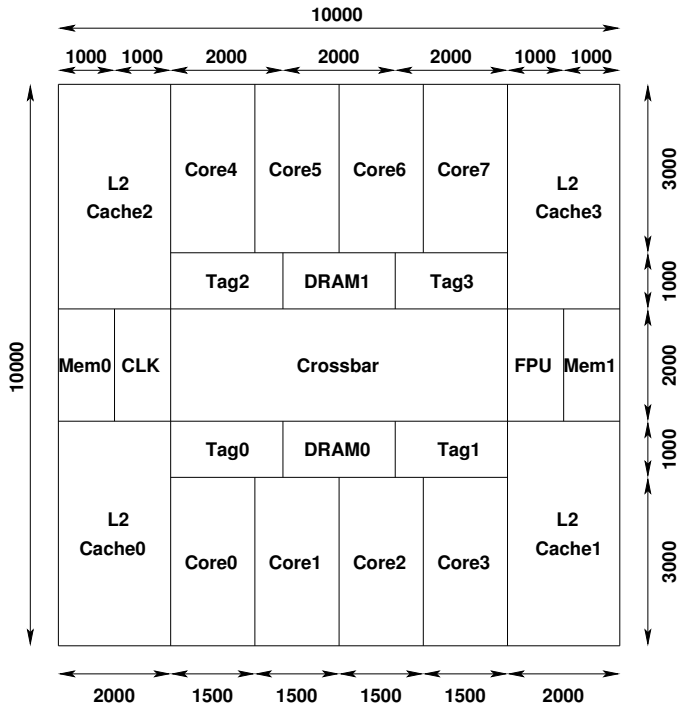

Figure 6.3: Floorplan of the Architecture

Node No.	Node Coord (x,y)	Processing Element	Node No.	Node Coord (x,y)	Processing Element
1	(1,1)	Cache0	14	(4,3)	FPU
2	(2,1)	Core0	15	(5,3)	Mem1
3	(3,1)	Core1	16	(1,4)	Cache2
4	(4,1)	Core2	17	(2,4)	Tag2
5	(5,1)	Core3	18	(3,4)	DRAM1
6	(1,2)	Cache0	19	(4,4)	Tag3
7	(2,2)	Tag0	20	(5,4)	Cache3
8	(3,2)	DRAM0	21	(1,5)	Cache2
9	(4,2)	Tag1	22	(2,5)	Core4
10	(5,2)	Cache1	23	(3,5)	Core5
11	(1,3)	Mem0	24	(4,5)	Core6
12	(2,3)	CLK	25	(5,5)	Core7
13	(3,3)	Crossbar			

Table 6.1: Node coordinates and their positions

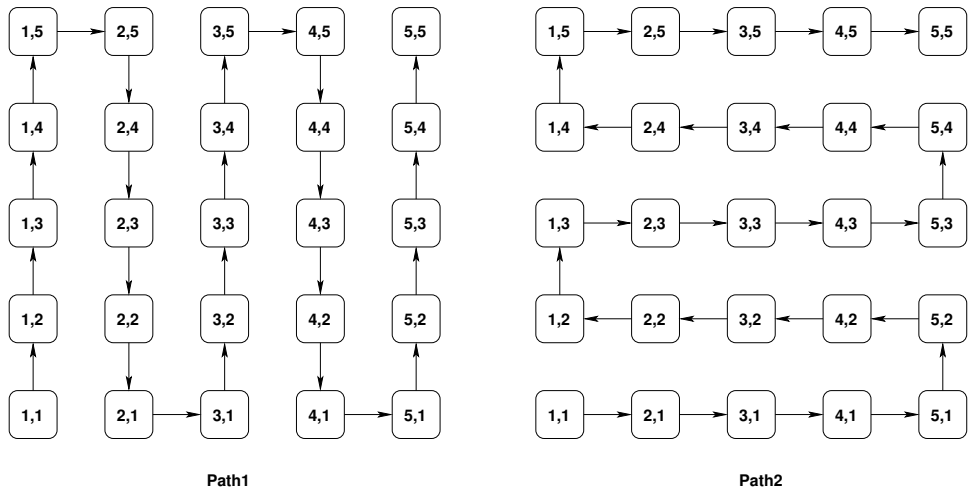

Figure 6.4: Path Configuration

of the nodes, the y-axis corresponds to y-coordinates of the nodes and we have a 5x5 grid network. The z-axis represents time in ms. The contour represents either temperature or modeling error and is described by a colormap. The datasets have 4 time slots and the plots have contour slices that correspond to the state of the system at the end of each time slot (25ms, 50ms, 75ms and 100ms on the z-axis).

The sensor positions (node coordinates associated with each processing element) are given in Table 6.1 and two path configurations that were used to analyze modeling errors are given in Figure 6.4. The paths are configured such that path1 has predominantly vertical links and hence, captures vertical gradients across the network and path2 has horizontal links and it captures horizontal gradients across the network. Based on how heat flows across the network for each dataset, the errors were analyzed and then, the errors characterized by plotting the bounds and analyzing the contribution of each term in the bounds equations to the change in bound values. The next subsection describes the four datasets along with analysis of how entity dynamics and data paths affect lumping and correlation errors. The subsequent subsection analyzes how the bounds that describe modeling errors change with entity dynamics. Eventually, the last subsection in this experiments section solves the optimization problem described by equations 6.25 to 6.39 that describes the procedure of optimized lumping based on cost function.

6.6.1 Analysis of Lumping and Correlation Errors

This subsection analyzes how data paths affect lumping and correlation errors for different scenarios produced by the datasets. The 4-D plots corresponding to temperature data is provided in this section. The plots that describe modeling errors are given in Appendix B. The subsections exclude the analysis of time synchronization errors as it was observed that these errors are path independent and highest at the positions of the heat sources. The maximum percentage error that was observed was 0.2% at dataset4, which is insignificant.

Dataset 1

For dataset1, core0 and core4 in figure 6.3 were heated up for a period of 100 ms (all time slots) by the same amount. Figure 6.5 shows the temperature values over the 25-node network and each slice in the figure corresponds to temperature value at the corresponding time instance (25ms, 50ms, 75ms and 100ms).

It can be observed that over time, since heat sources are stationary at core0 (2,1) and core4 (2,5), area covered by those cores and the associated caches heat up. The associated caches are cache0 (1,1) for core0 and cache2 (1,5) for core4. The temperature values local to those processing elements are hotter than the other parts, but the heat propagates over the network due to heat transfer gradients. As a result, the nodes on the right part of the network at 100ms are warmer compared to their values at 25ms, even though they are not located at the heat spots.

Analyzing Lumping Errors for Dataset 1:

Figure B.1 shows the percentage lumping error at each point in the 5x5 network when path 1 is used and figure B.2 shows the percentage lumping error when path2 is used. When path1 is used, percentage error at nodes (2,1), (2,2), (2,4) and (2,5) are very high for all 4 time slots compared to other nodes in the network. Node (2,1) is located at core0 and node (2,5) is located at core4. Analyzing those parts of the network, path1 mainly has vertical links. Hence the two segments (2,2), (2,1), (3,1) and (1,5),(2,5),(2,4) lie in the areas where percentage error is very large. Lumping the node (2,1) would establish a new link between (2,2),(3,1) and the model parameters S_c and G_r are accordingly adjusted. It is observed that due to very high temperature value at core0 (2,1), the updated G_r values after lumping (2,1) and/or (2,2) would result in loss of crucial information. Since the S_c values are also removed, the high

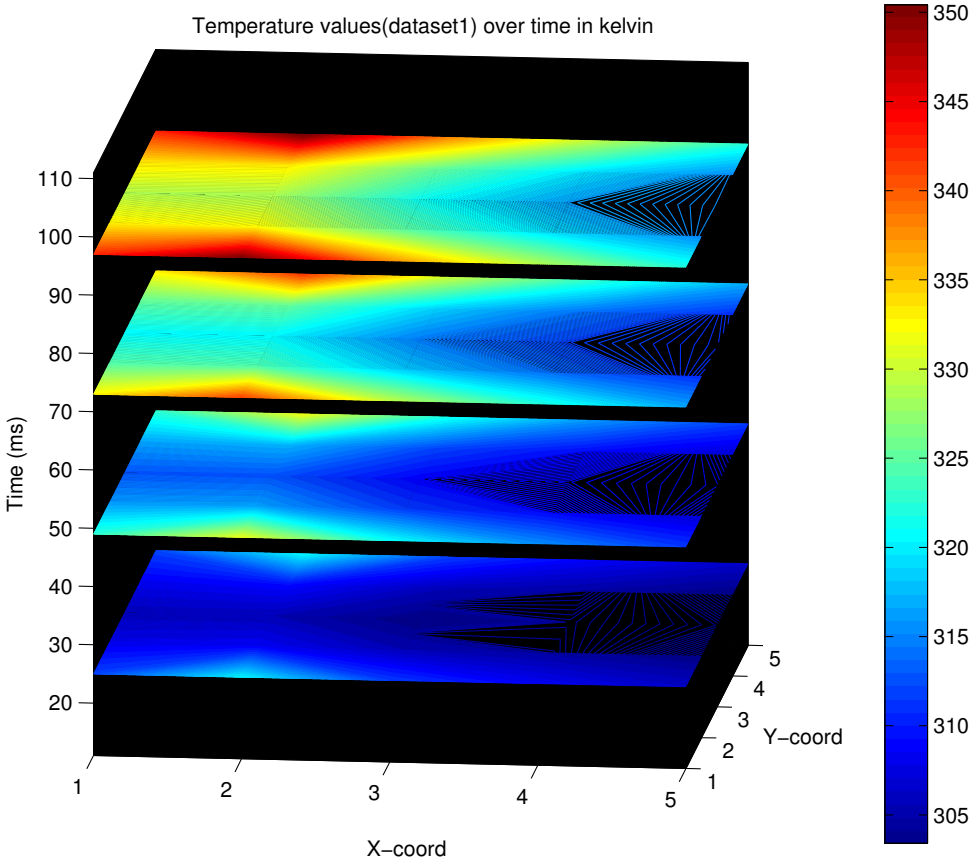

Figure 6.5: Temperature values for dataset 1

heat values stored at these nodes would not reach the target point. Similarly, error increases when you lump either node (2,5) or node (2,4) due to the lost information at node (2,5) which lies in core4. Error is computed by computing the difference between the actual temperature value at the node and the temperature value estimated after lumping the node and updating the coefficients. Due to the nature of path1 having vertical links, the plots in figure3 shows that percentage error at the heating cores and their vertical neighbors is very high.

The same reasoning occurs for path2 but in this case, the paths are pre-dominantly horizontal. It is interesting to observe that more number of nodes are affected near core4 (2,5). The path segment that is affected is (3,5),(2,5),(1,5),(1,4),(2,4) with high percentage errors at (3,5),(2,5),(1,4),(2,4) due to the high loss of information along the path. For node (2,1), error is high between nodes (2,1) and (3,1) due to the fact that (2,1) is located at the heated core0.

Analyzing Correlation Errors for Dataset 1:

Figure B.3 shows the percentage correlation error at each point in the 5x5 network when path 1 is used and Figure B.4 shows the percentage correlation error when path2 is used. It can be observed that path1 is a lot more efficient compared to path2 for this particular dataset.

The reason why path1 produces lower values of correlation errors than path2 is due to the nature of the paths and the location of the two heat sources. It is observed that the two nodes located at the core have only one missing link at both nodes. However, the nodes especially on the left part of the network have very high temperature gradients along the vertical links. The two cores get heated up and heat dissipation and gradients are such that the temperature values at these cores become very high, but the non-core and non-cache elements that are located at vertical links with respect to the cores do not heat up as much. Hence, the temperature gradients at the vertical links of the cores are very high due to this difference and produces higher correlation errors if these nodes are not linked in the path. Hence, path1 performs better compared to path2 since it is configured to have predominantly vertical links. Only node (4,3) has high correlation errors when path1 is used since it is located at the FPU and it heats up more compared to the other processing elements nearby causing more gradients along the horizontal direction (refer to Figure 6.3 and Table 6.1 to determine the positions of FPU and its neighbors). For nodes located at row 2 (y-axis = 2), the gradients are higher along the bottom link due to the presence of a

hot core (core0) at row0. Similarly, row4 has high gradients along the top link due to the presence of core4 (located at row5) which is hot and row3 has both vertical and horizontal links with high gradients. This behavior is more visible at the left side of the network since both core0 and core4 are located on the left side (x-axis = 2).

Dataset 2

Dataset2 has moving heat sources. Core0 and core7 in Figure 6.3 were heated up for the first time slot (0-25ms). Then the heat sources move to core1 and core6 in the second time slot (25-50ms). For the third slot (50-75ms), the heat sources are located at core2 and core5 and eventually, core3 and core4 get heated up in the fourth time slot (75-100ms). Figure 6.6 shows the temperature values over the 25-node network and each contour slice in the figure corresponds to temperature values at the end of each time slot (25ms, 50ms, 75ms and 100ms).

Nodes (2,1) and (5,5) display higher temperature values at 25ms since they are located at core0 and core7. For the second time slot, nodes (3,1) located at core1 and (4,5) located at core6 display higher temperature values but the effects of the previous heat sources can still can be seen at those points. For the third time slot, nodes (4,1) and (3,5) heat up due to their locations at core2 and core5 and the residual heat at the previous heat positions can be seen. Eventually, the fourth time slot that ranges from 75-100ms heats up core3 and core4, i.e. nodes (5,1) and (2,5) in the network, but due to the fact that both heat sources moved along the x-direction to the other end of the network, temperature is fairly high at the cores through which the heat sources passed, but obviously temperature is the higher at the nodes local to the current heating cores. Previously, for dataset1, since the cores were stationary, heating was only local to those areas. Hence, max temperature is higher at 100ms for dataset1, although heat is more distributed over the network for dataset2.

Analyzing Lumping Errors for Dataset 2:

It is observed from figures B.5 and B.6 that lumping error is higher local to the position of the heat source and the reason is same as the lumping errors that occur for dataset1. It is, however, observed that path2 performs better than path1 as time passes since the other nodes in the network also display 1.5% - 2% errors when path1 is used while it is less than 1% for those nodes (many of them have close to 0% error) when path2 is used. This behavior is most visible at the fourth contour slice that corresponds to the end of the last time slot. This is because of the residual heat at the

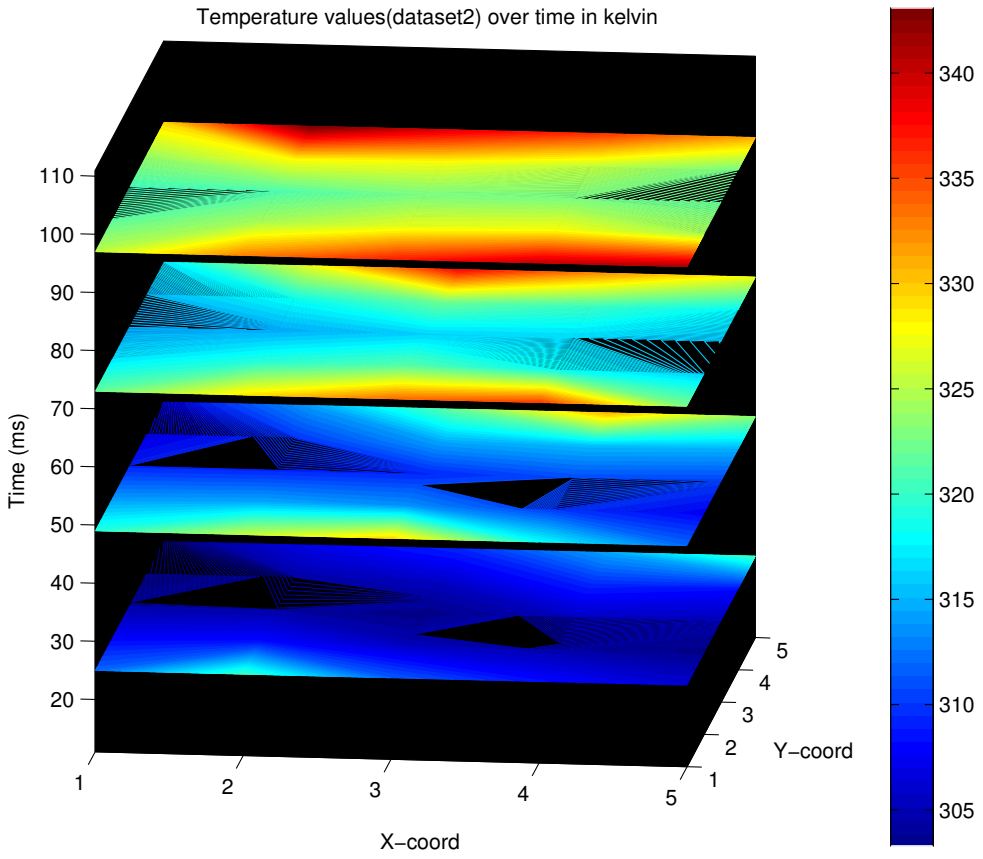

Figure 6.6: Temperature values for dataset 2

previously heated cores resulting in lower temperature gradients along the horizontal direction. This phenomenon is observed along the red areas of the fourth contourslice in Figure 6.6 that shows temperature plots. The gradients are higher in the vertical direction closer to the heat spots (red to yellow areas in Figure 6.6).

When you lump a node with higher gradients with respect to neighbors, some amount of information is lost. Since path2 mostly uses horizontal links, error is close to zero at the nodes that are away from the hot spots due to smaller gradients and even local to the heat spots, the error is much lower compared to the results produced by path1. Segment (5,1), (5,2), i.e. the lower-right part of the network displays high lumping errors when path2 is used for the last time slot. The segment, being a vertical link, lies between the yellow and the red regions in Figure 6.6 causing an increase in lumping errors. This phenomenon is observed only during the last time slot because the heat source moves to core3, i.e. node (5,1) during this time. Hence, dataset2 provides us with the additional insight over dataset1 that the decision to lump a node depends not only on its absolute temperature value (position of heat sources), but also its relationship with respect to neighbor nodes (in terms of gradients).

Analyzing Correlation Errors for Dataset 2:

Figure B.7 shows the correlation errors for dataset2 when path1 is used and Figure B.8 shows the correlation errors for dataset2 when path2 is used. Similar to dataset1, path1 performs better than path2 in minimizing correlation errors. Especially at 100ms, the end of the fourth time slot, correlation error is almost zero except for a couple of nodes at the bottom-left part of the network. On the other hand, except for 3 nodes in the network, correlation error in most part of the network is 90% when path2 is used at 100ms. For dataset1, correlation errors reduced as you moved away from the stationary heat sources but in this case, both heat sources move along the horizontal direction at top and bottom part of the network causing high gradients along the vertical axis throughout the network. This results in a significant increase in correlation errors when path2 is used, but path1 reduces these errors to almost zero due to the fact that it predominantly has vertical links in its path, thus capturing all correlations between nodes that have very high gradients, e.g. the vertical links cover the gradients represented by the red and yellow areas in Figure 6.6.

The high correlation errors that occur at (1,1) when path1 is used is due to the fact that its right neighbor (2,1) is a core and due to the fact that the link between those nodes is absent, the gradient between those nodes is high. On the other hand,

that link exists when path2 is used and hence, error is minimal for that node. The two blue spots that always exist at nodes (5,1) and (1,5) are due to the fact that those nodes do not have any missing links for either paths. This condition is true for all datasets.

Dataset 3

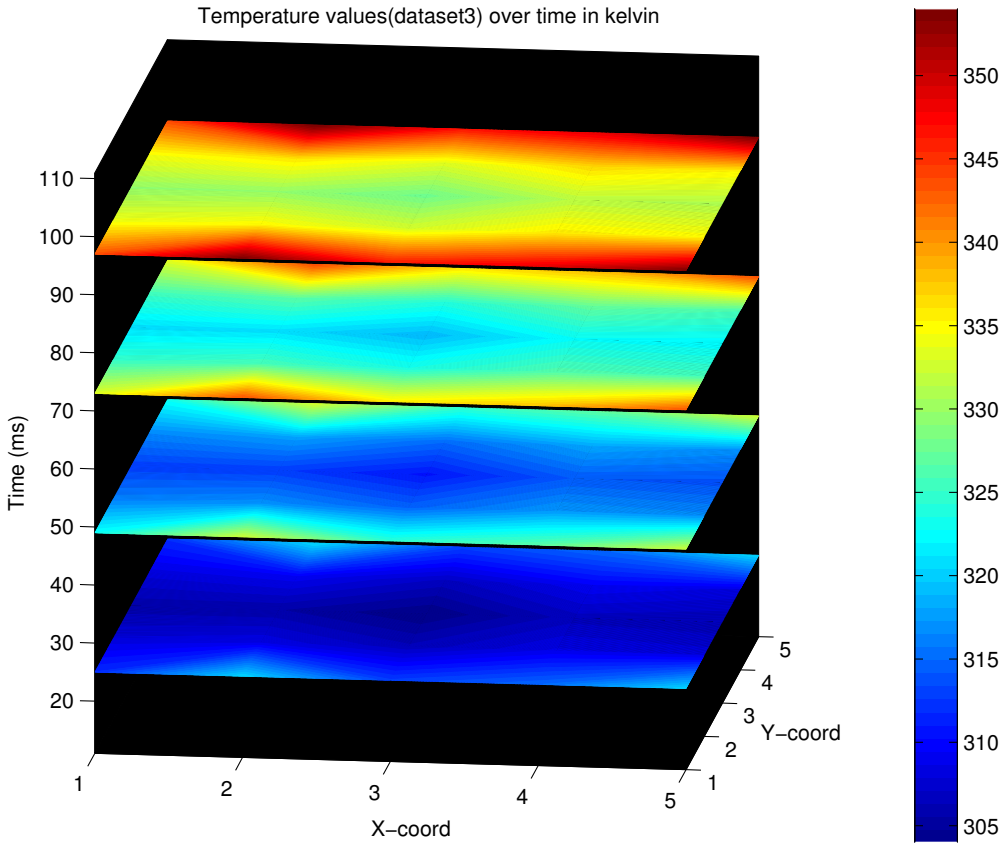

Figure 6.7: Temperature values for dataset3

Dataset3 has 4 stationary heat sources located at core0, core3, core4 and core7. These heat sources are heated by the same amount during all 4 time slots over the

period of 100 ms.

The four cores located at (2,1), (5,1), (2,5), (5,5) are heated over a period of 100ms. It is observed from Figure 6.7 that the nodes in the network gradually experience an increase in temperature due to gradients. At 100ms, the nodes along the first row (y = 1) and the last row (y = 5) heat to very high temperatures compared to the other nodes. In other words, heat transfer was maximum along these two rows since there was a heat source on each end causing the red lines at those two rows in the fourth contour slice in Figure 6.7. Due to the presence of 4 heat sources at the corners, overall temperature is definitely high compared to dataset1, which contains the blue (colder) regions in the right part of the network, since the 2 cores were located on the left part.

Analyzing Lumping Errors for Dataset 3:

The behavior is consistent with the previous 2 cases for path1 where percentage error is high local to the heat spots as shown in Figure B.9. However, Figure B.10 shows that when path2 is used, lumping errors are much lower even at the heat spots. It is higher at the other heat spots compared to the rest of the network but much lower than the results observed for dataset1 and dataset2. This is due to the fact discussed in the temperature analysis that the gradients are very low along the x-axis at the first and last rows compared to the previous datasets. For example, at y = 1, nodes (2,1) and (5,1) have heat sources for this dataset while there is only one heat source at (2,1) for dataset1. The same path segment (2,1), (3,1) is used for both datasets when path2 is selected. For dataset1, you observe high lumping errors at nodes (2,1) and (3,1) while the lumping errors at the same time instances are 50% lower at those nodes for dataset3, the only difference being that (5,1) also has a heat source. The gradient effects from (5,1) increases the temperature at (3,1) resulting in lower temperature gradients between (2,1) and (3,1).

When path1 is used, the segments are such that nodes neighboring to the heat spots (yellow region in the fourth contourslice) are at a lower temperature values compared to the heated nodes (red region in the fourth contourslice) indicating higher gradients and hence, resulting in higher lumping errors. Similar situation occurs at segment (5,1), (5,2), i.e. the lower-right part of the network when path2 is used. The segment being a vertical link lies between the yellow and the red regions causing an increase in lumping errors. This behavior was also observed for dataset2.

Analyzing Correlation Errors for Dataset 3:

Figures B.11 and B.12 re-establish the fact that path segments should be selected along links that are very high gradients to reduce loss of correlations between nodes. The presence of four heat sources generates more gradients in the network resulting in higher percentage errors when path2 is selected compared to the previous datasets with 2 cores. Path2 definitely performed better, especially in the right part of the network for the initial time instances for dataset1. It also performed better for dataset2 for the first time slot when the 2 heat sources had not started moving. Since this dataset contains 4 heat sources at four corners in the network, path2 performs worse for all 4 time slots due to the nature of the path segments defined in the path. As expected, high temperature gradients at the end of the fourth time slot cause very high correlation errors at the fourth contourslice in figure B.12.

Dataset 4

Dataset4 has 4 heat sources initially located (in the first time slot) at core0, core3, core4 and core7 just like in dataset3. But they move after each time slot. Referring to the first contourslice (end of first time slot) in Figure 6.8, temperatures at nodes where the cores are located have higher values compared to the other nodes. The corresponding nodes are (2,1), (5,1), (2,5), (5,5).

For the second time slot, the heat source at core0 moves to core1, the heat source at core3 moves to core2, the heat source at core4 moves to core5 and the heat source at core7 moves to core6. Hence, the heat sources are now located at (3,1), (4,1), (3,5), (4,5). It can be seen that temperature values are higher at the those nodes (central part of the first and last rows). For the third time slot, the heat sources located at core1 and core2 swap their positions and the heat sources located at core5 and core6 swap their positions. Hence, the same core elements have the heat sources, the only difference being that they were swapped. It can be observed from the third contourslice that the temperature at the central part of the first and the last rows are higher compared to the previous slice. For the fourth time slot, the heat source at core 1 moves to core0, the heat source at core2 moves to core3, the heat source at core5 moves to core4 and the heat source at core6 moves to core7. Hence, the same core elements that were heated during the first time slot get heated again (four corners of the chip), but with the heat sources swapped. The fourth contourslice shows the temperature at the end of the last time slot and the first and the last row elements are hot due to the moving nature of the heat sources and their residual effects. The

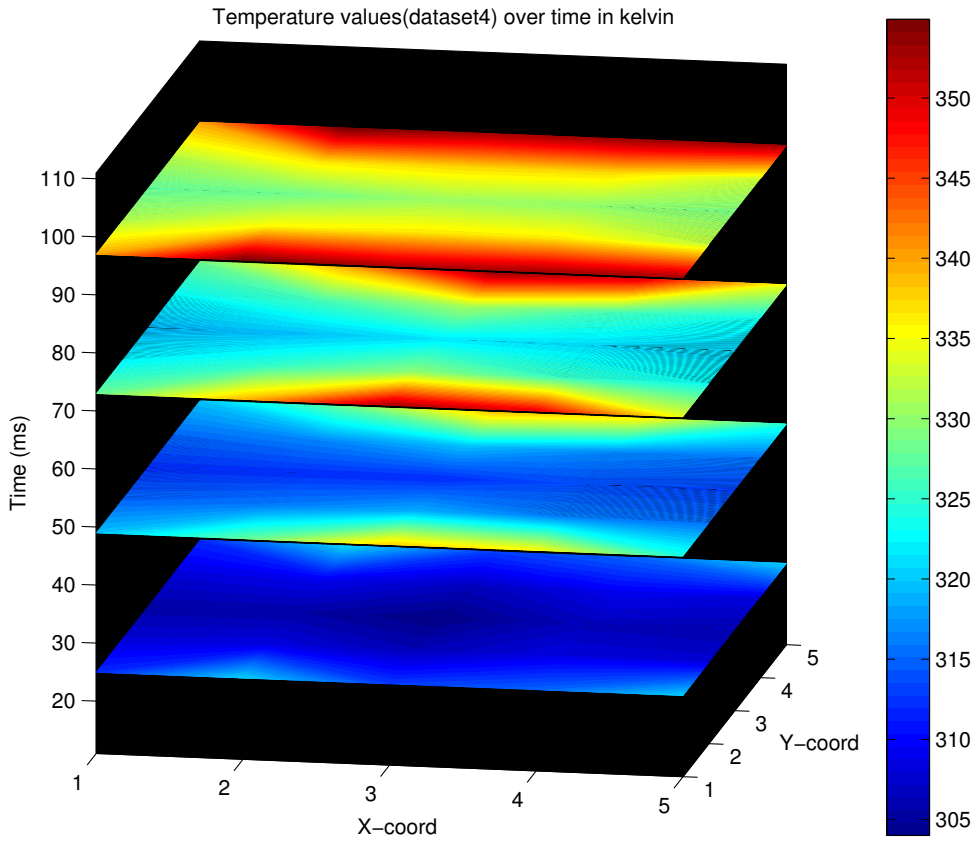

Figure 6.8: Temperature values for dataset4

fourth contourslice for dataset4 in Figure 6.8 may look the same as the contourslice for dataset3 in Figure 6.7, but note that the nodes at the center are more red (hotter) for dataset4 compared to dataset3 due to the moving nature of the heat sources.

Analyzing Lumping Errors for Dataset 4:

The nature of the moving heat sources can be observed in Figure B.13 since lumping errors are very high local to the heat spots. Since path1 is used, the nodes located at the heat spots and the vertical neighbors display high lumping errors. Expectedly, for the second and the third time slots, the central nodes of the first and last rows produce very high lumping errors along with their vertical neighbors due to the presence of two heat sources at those points. When path2 is used, the lumping errors produced by this dataset shown in Figure B.14 is very similar compared to dataset3, the only difference being that the reason why those similar gradients are produced (the reason being the moving nature of the heat source). It can be observed that it is a bit lower compared to dataset3 since all nodes along the rows are heated by the moving cores resulting in lower gradients between those nodes. Another difference that was observed was that the vertical segment at the bottom-right part of the network (segment (5,1),(5,2)) produces higher errors for time slot 1 and time slot 4 since the heat source moves during the other time slots. For dataset3, that part of the network always produced high lumping errors since the heat sources were stationary.

Analyzing Correlation Errors for Dataset 4:

For this dataset, the plots in Figure B.15 show that path1 produces correlation errors very similar to dataset3, but for the difference that the effects of the FPU at the center of the network (this effect was discussed while analyzing dataset1) is visible at the initial time instances, but that effect is not to be seen during the later time instances for this dataset. In fact, at the end of the fourth time slot, this effect completely vanishes and the correlation errors around that point is close to zero. The reason is that as the heat sources move, they transfer heat over the network. This aspect dominates the effects of the FPU.

The plots in Figure B.16 show that path2 produces much higher correlation errors compared to dataset3 due to the moving nature of the cores. The fact that the cores moved along the x-direction created a very high gradient with respect to the vertical links for all nodes. We can see a larger difference between the red and yellow zones from fourth slice in Figure 6.8 (dataset4) compared to the same plot in Figure 6.7 (dataset3) indicating the larger gradients produced by dataset4. As a result, this

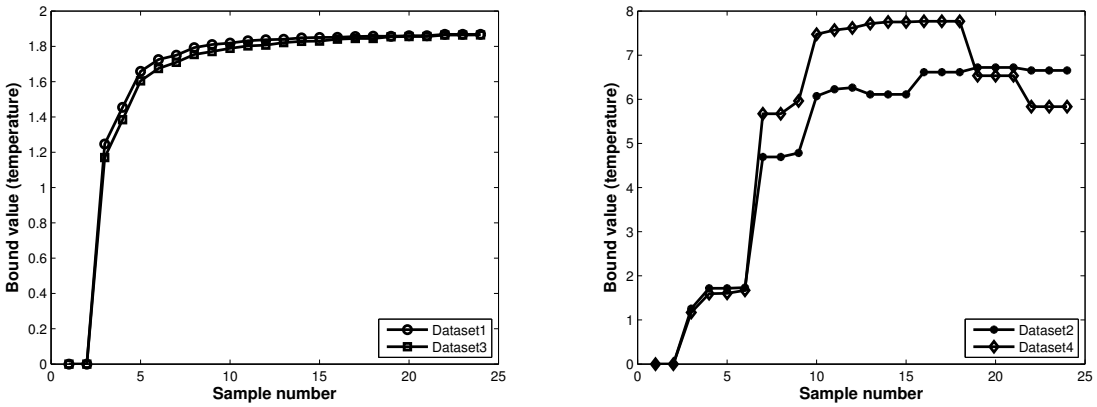

Figure 6.9: Bounds for lumping errors at core 0

dataset produced the highest amount of correlation errors when path2 is used. Hence, we observe dark red areas in most part of the network at the end of time slot 4.

6.6.2 Analysis of Bounds for Lumping Errors

This subsection analyzes the bounds for lumping errors given by equations 6.8, 6.9, 6.10. The bound values at core0, core1, core2 and core4 are analyzed for path2 and they provide good insight on the factors that might change the lumping errors and aid in the process of deciding whether to lump a node depending on current conditions, i.e. the dynamics of the monitored entities in the vicinity of the node. Each of the four plots given by Figures 6.9, 6.10, 6.11 and 6.12 have two plots where the plots on the left hand side cover bound values from the fixed datasets (i.e. dataset1 and dataset3) and the plots on the right side cover bound values from the moving datasets (i.e. dataset2 and dataset4). It is observed that the bound values for all cores reach steady state after the initial few samples for fixed datasets. The bound values are sensing readings, i.e. temperature values in this case.

Core0

Analyzing core0, i.e. node (2,1) for dataset1 and dataset3 given by Figure 6.9, a heat source is always present at this core. Hence, it produces almost exactly the same bound values for both datasets, the bounds for dataset1 being slightly higher due to the difference in gradients with respect to its neighbors. Dataset3 has one more heat

source at core3 (5,1), resulting in lower gradients. For these datasets, since the heat source is static at core0 and the power dissipation does not change over time, the $\ddot{Y}_{x_{MAX}}$ value corresponding to the first term in the equation 6.9 is at its maximum at this node and remains the same for the other samples. Hence, it reaches a steady state after 6 samples and remains fairly constant over time. The time taken after sample 3 through sample 6 to reach steady state is due to the increase in S_c value between samples 3 and 4, causing an increase in the bound value by 13% and the steady decrease in $\dot{E}_{comp_{MIN}}$ value in equation 6.10 between samples 3 and 6, causing an increase in the bound value by 14.5% at sample 5 and 4% at sample 6. After this, all the terms remain fairly constant over time.

For dataset2, the heat source is present at the first time slot and then it moves along the horizontal direction at the subsequent time slots. It reaches core3 (5,1) at the fourth time slot. In case of dataset4, the heat source present at core3 reaches core0 at the fourth time slot. It is observed that the bound values are consistent with the static datasets for the first time slot for both dataset2 and dataset4. However, the heat source moves at the sixth sample and hence, the bound values increase again. There is a sudden increase in the value of gamma and the term $\dot{Y}_{B-x_{MAX}}$ in equation 6.9 since the node moves to node B, the corresponding link for node (2,1) located at core0. The term gamma increases by a large amount due to the high difference in gradients between the two links. This results in the decrease of the value of κ, but this effect is mitigated by the increase in $\dot{Y}_{B-x_{MAX}}$. Hence, an increase in bound value is observed at sample 7 by 171% for dataset2 and 240% for dataset4. For dataset4, the value of $\dot{Y}_{B-x_{MAX}}$ is much higher due to the effects of another heat source. At sample 10, the gradient effects at the link between nodes 2 and 3, i.e. link $B-X$ changes its state due to heat transfer over time between the nodes. This affects the G_r values and the value of γ decreases resulting in an increase in κ. As a result, we could observe an increase in the bound value by 25% for both datasets. Hence, the effects of κ is very less compared to the effects of the $\dot{Y}_{B-x_{MAX}}$ and $\ddot{Y}_{x_{MAX}}$ values. For dataset4, the bounds reaches steady state until sample 18 since two heat sources are present at core1 and core2 during this time. For the fourth time slot, the heat sources move back to core0 and core3. The bound values decrease for dataset4 during this time by 16% at the start of the fourth time slot (sample 18) due to the difference in gradients caused by the moving of the heat sources. The $\dot{Y}_{B-x_{MAX}}$ and $\ddot{Y}_{x_{MAX}}$ terms do not change at this point because the heat sources have already traversed all the

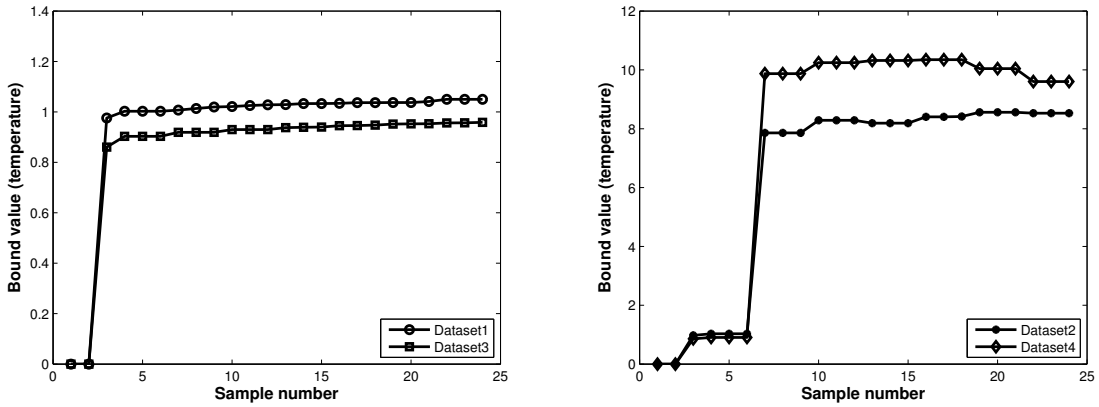

Figure 6.10: Bounds for lumping errors at core 1

cores along the x-axis, and the gradients are not high enough to change them, but are sufficient to change κ resulting in the 16% decrease. Similar to the second time slot, the gradient effects at the links change over time due to heat transfer caused by the moving cores at the fourth time slot and the bound value decreases in the middle of the fourth time slot (sample 22) by 11%.

Core1

Bounds for core1 are given in Figure 6.10. Compared to core0, bounds for core1 when dataset1 and dataset3 reach steady state more quickly and at steady state, they are approximately 41% lower since no heat source is present at this node for those datasets. In other words, $\ddot{Y}_{x_{MAX}}$ increases depending on energy received from the neighboring core and hence, is lower. For both dataset2 and dataset4, the heat source reaches this core at the second time slot. Hence, a huge increase in bound value (663% for dataset2 and 988% for dataset4) is observed at sample 7. This is due to the significant increase in $\ddot{Y}_{x_{MAX}}$ values due to the moving of the heat source into this core. For the later time instances, few increases and decreases in bound values were observed due to the gradient effects explained for core 0.

Core2

Bounds for core2 are given in Figure 6.11. When dataset1 is used for core2, it is further away from the heat source at core0 and hence, the bounds reach steady state

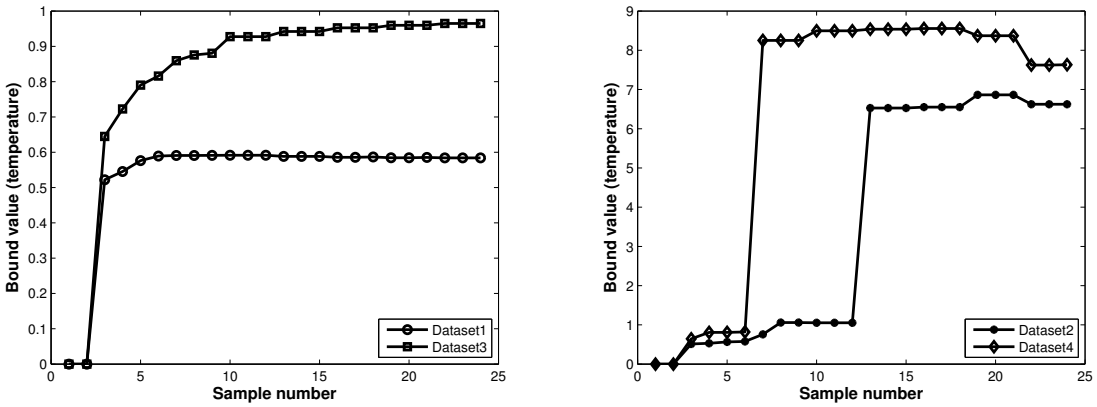

Figure 6.11: Bounds for lumping errors at core 2

almost immediately and the bound value at steady state is 96% lower than core0. For dataset3, the bounds for this core is similar to core1 since both cores have a neighboring core with a heat source. Dataset2 behaves similar to core1, the difference being that the huge increase in bound value (522%) is time shifted by a factor of 25ms (6 samples). This is because the heat sources moves into this core during the third time slot compared to the second time slot in the case of core1. For dataset4, the bound value increases at the same time compared to core1 since heat source from core3 moves to this core during the second time slot and the increase in bound value (906%) is also similar to core1.

Core4

The bound values at core4 are given in Figure 6.12. It behaves very similar to core0 for dataset1 and dataset3 since the conditions are very similar. However, for dataset2, the behavior is similar to cores 1 and 2, the difference being that the huge increase in bound value is observed at the start of the fourth time slot, when the heat source moves to this core.

Characterizing Lumping Errors

Combining the analysis of the increase in lumping errors from the 4D plots and the contribution of each term in the bound equation due to the effects of the moving entities from the bound plots, it was concluded that the lumping error increases

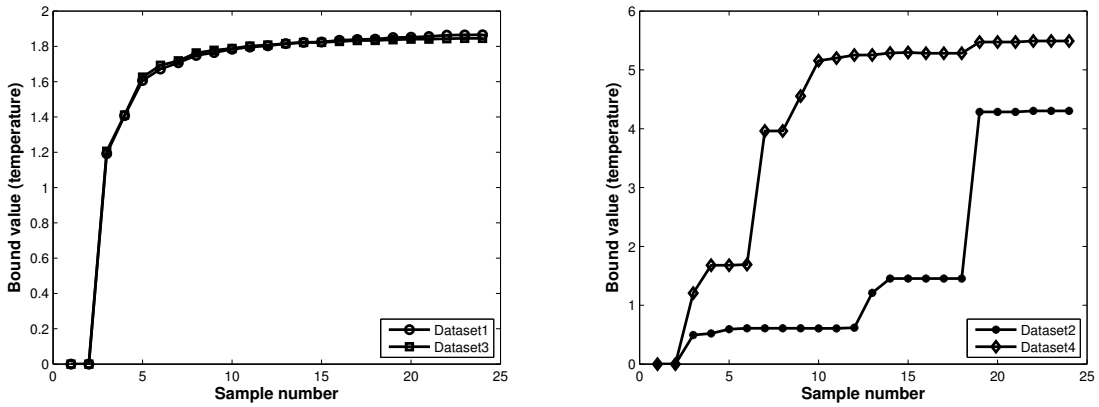

Figure 6.12: Bounds for lumping errors at core 4

significantly when the heat source moves to the vicinity of a node (significant increase in $\ddot{Y}_x$ values, the first term in the bound equation) and when the gradient is very large (increase in κ and the second and/or third term, $\dot{Y}_{A-x}$ and/or $\dot{Y}_{B-x}$ values in the bound equation). The change in the value of κ occurs due to the change in G_r values, but the effect on the bounds is insignificant compared to the other factors.

Based on the above analysis, we can add meaning to the equation that describes the bounds for lumping errors. The first term indicates the increase in information when a heat source is present in the core. The second and the third term indicate the gradients with respect to neighboring nodes in the path. The equation is formulated such that higher the increase in temperature over time and higher the gradients, higher would be the lumping errors due to loss of information at the node and the gradient effects with respect to the neighbors. The fourth term indicates the loss of information due to the change in energy sinks at the node that is lumped.

6.6.3 Analysis of Bounds for Correlation Errors

This subsection analyzes the bounds for correlation errors given by equations 6.18, 6.19, 6.20. The bound values at core0, core1, core2 and core4 are analyzed for path2 and they provide good insight on the factors that might change the correlation errors. Each of the four plots given by Figures 6.13, 6.14, 6.15 and 6.16 have two plots where the plots on the left hand side cover bound values from the fixed datasets (i.e. dataset1 and dataset3) and the plots on the right side cover bound values from the moving

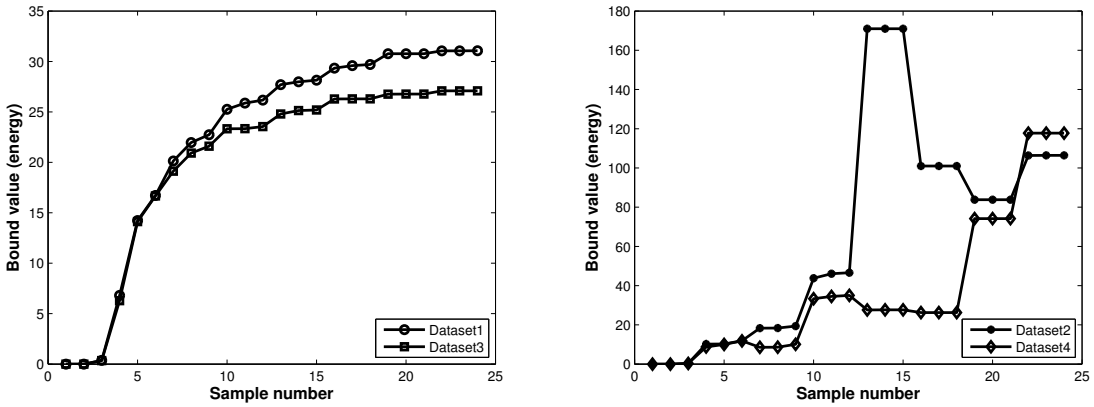

Figure 6.13: Bounds for correlation errors at core 0

datasets (i.e. dataset2 and dataset4). The bound values are in terms of energy since correlation errors correspond to energy at the missing links. In this case, we measure energy flux since we measure temperature, i.e. heat.

Core0

The plots that analyze bounds for core0 are given in Figure 6.13. When dataset1 and dataset3 are used, the heat source at core0 remains static for all time slots. Hence, the $\ddot{Y}_x$ value is initially high until it reaches steady state approximately at the end of the first time slot. At this point, the maximum value of $\ddot{Y}_{x_{MAX}}$ for all time slots is reached. Once steady state is reached, the gradient values with respect to the neighboring nodes keep decreasing over time, decreasing the $\dot{Y}_{B-x_{MIN}}$. This results in the increase in bounds for correlation errors. This behavior is consistent with the analysis of correlation errors using the 4-D plots. A decrease in $\dot{Y}_{A-x}$ and $\dot{Y}_{B-x}$ values increases the energy difference between the rise of energy at the state variable $\ddot{Y}_x$ and the energy due to gradients with respect to the path's established links. This difference in energy corresponds to the missed correlation (energy) with respect to the missing links. Hence, once steady state is reached for dataset1 and dataset3, the gradients keep decreasing and there is a steady increase in bound values of correlation errors.

For dataset2 and dataset4, the moving of the heat sources at the end of first time slot creates the required gradients that keeps the bound values low until the third

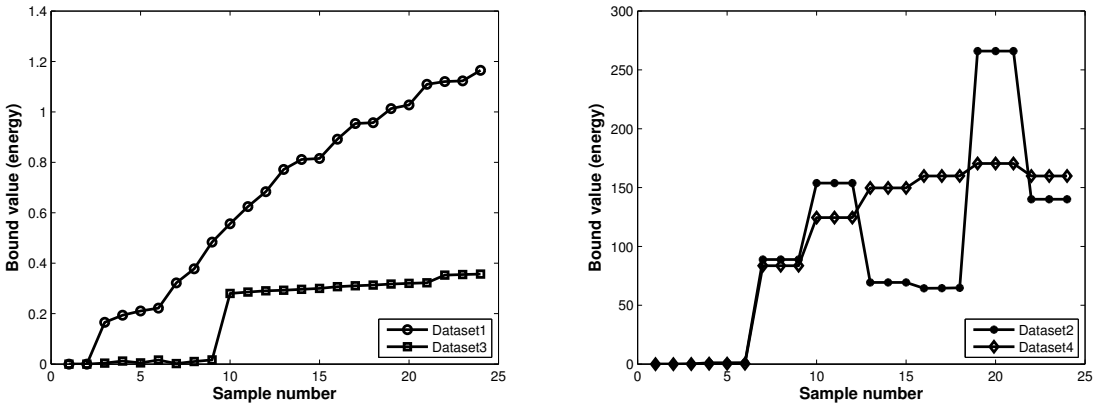

Figure 6.14: Bounds for correlation errors at core 1

time slot is reached. The heat source is present in the first time slot at core0 and the same heat source moves to core1 at the second time slot, keeping the high gradients between these cores. At the third time slot, the heat source moves to core2, which is furthest away from core0, significantly decreasing the gradients that produces a spike in the bound value by 260% between samples 12 and 13. After the heat source moves to core2, the relative gradient values change as it reaches equilibrium and the value of γ increases, resulting in a decrease of the bound values by 40%. When dataset4 is used, the heat source is present at core0 during the first time slot, and a heat source is present at core1 during the second and third time slots. This keeps the gradient high between core0 and core1, resulting in low bound values. However, the heat source moves from core1 to core0 during the fourth time slot, resulting in lower gradients between the cores as the increase in temperature at core0 brings its temperature closer to the temperature of core1. Hence, the increase in bound value of 203% is observed between samples 18 and 19.

Core1

The plots that analyze bounds for core1 are given in Figure 6.14. Since there is no heat source present at core1 for dataset1 and dataset3, the range of bound values (0 to 1.2) is significantly lower compared to the range of bound values at core0 (0 to 32). For dataset2 and dataset4, the heat source moves from core0 to core1 causing a huge increase in $\ddot{Y}_{x_{MAX}}$ at core1. This results in a sharp increase in bound value from

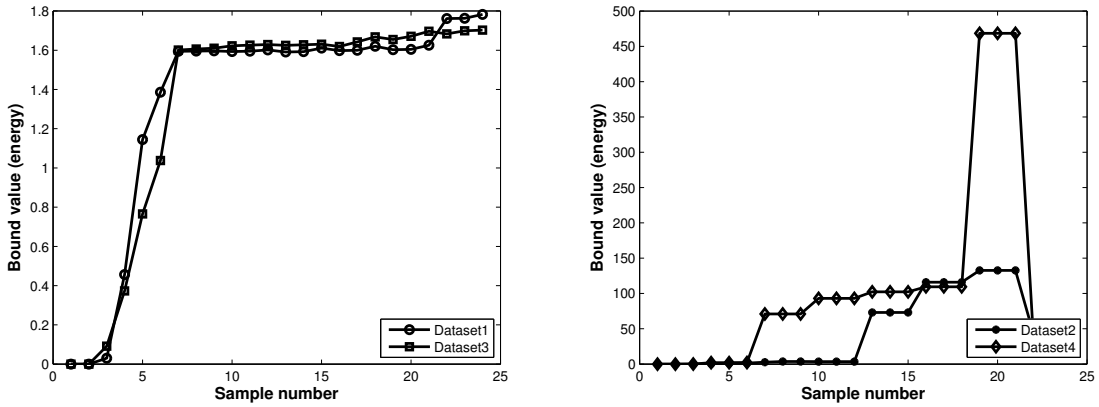

Figure 6.15: Bounds for correlation errors at core 2

a flux value of 1.18 to 89 (i.e. more than 7000%) between samples 6 and 7. At the middle of the second time slot, the gradients between the two cores decreases as the temperature of core1 gets closer to core0 resulting in an increase in bound value of 73% for dataset2 and 48% for dataset4 between samples 10 and 11. The increase of bound value by 315% at the start of fourth time slot due to the moving of heat source (to core3) further away from the core (core1) and the subsequent decrease by 47% due to the change in γ is similar in concept to the drastic changes of bound values observed during the third time slot at core0.

Core2

The plots that analyze bounds for core2 are given in Figure 6.15. For this core, bound values are very low for dataset1 and dataset3 similar to core1 due to the fact that no heat source is incident on this core at any point of time. For dataset2, the bound values behaves similar to the bound values at core1, the only difference being that the increase from a very low value of 3.5 to 73 occurs at the third time slot when the heat source moves to this core, resulting in a sudden increase in $\ddot{Y}_{x_{MAX}}$. In case of core1, it moves during the second time slot causing the spike in bound values. For dataset4, the same increase is observed during the second time slot when the heat source moves from core3 to core2. In addition, there is also a 450% increase of bound values at the start of the fourth time slot when the heat source moves away from this core to core3. Interestingly, the temperature value at the node decreased and caused

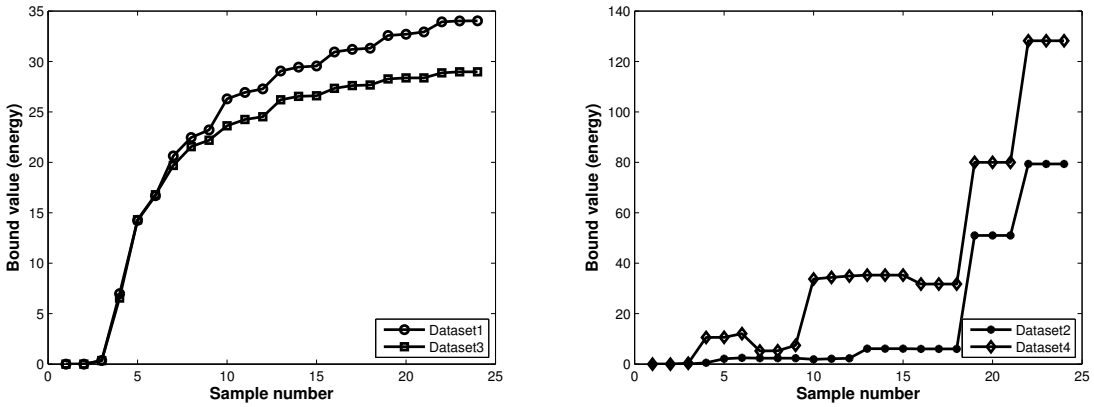

Figure 6.16: Bounds for correlation errors at core 4

the value of bound value to spike since the terms in equations 6.19 and 6.20 ended up being added instead of subtracted due to opposite signs of the values that result from this behavior. The temperature again starts increasing after a couple of samples and during this time instance, the term given by equation 6.19 again computes a positive value. In addition, steady state condition between the gradients and the $\ddot{Y}_{x_{MAX}}$ term caused the bound value to decrease significantly. This phenomenon is observed momentarily even at other nodes when a heat source leaves the node and enters its neighbor, which happens to be the next node in the path.

Core4

The plots that analyze bounds for core4 are given in Figure 6.16. The bound values at core4 are very similar to core0 for dataset1, dataset3 and dataset4. This is due to similar relative positions with respect to heat sources. However, for dataset2, the heat source moves towards core4 from core7 as opposed to the phenomenon at core0 where the heat source moves from core0 to core3. Hence, the bound values are low for the initial three time slots and increases significantly at the start of the fourth time slot (sample 18) when the heat source moves to this core. The presence of a heat source over time causes a decrease in gradient with respect to core5 resulting a further increase in the middle of the time slot (sample 22).

Path	Unoptimized				Optimized				
Conf.	BW	% Loss	Avg Delay	Avg Error	Opt Ratio	%Loss	%Lump Loss	Avg Delay	Avg Error
1	2	3	4	5	6	7	8	9	10
Path 1	BW 1	43.75	18.65	7.36	Opt 1	59.41	24.11	23.05	6.57
	BW 2	56.25	22.22	6.89	Opt 2	64.33	39.77	21.82	6.19
	BW 3	63.54	30.62	6.92	Opt 3	61.78	57.06	23.06	4.98
Path 2	BW 1	43.75	17.78	6.73	Opt 1	60.6	23.63	25.30	6.87
	BW 2	56.25	22.45	7.08	Opt 2	69.04	49.4	27.67	6.04
	BW 3	63.54	30.82	6.5	Opt 3	72.63	65.78	26.48	5.02

Table 6.2: Comparison of results: Dataset with static heat sources

Characterizing Correlation Errors

Combining the analysis of the increase in correlation errors from the 4D plots and the contribution of each term in the bound equation due to the effects of the moving entities from the bound plots, it was concluded that the correlation error increases significantly when the heat source moves to the vicinity of a node (significant increase in $\ddot{Y}_x$ values) and when the gradient is very low (decrease in $\dot{Y}_{A-x}$ and/or $\dot{Y}_{B-x}$ values in the bound equation). A decrease in gradient values increases the difference in the energy stored at the node and the gradient energies at the path links. This difference in energy corresponds to the total energy due to missing links.

6.6.4 Optimized Distributed Data Lumping

The procedure of optimized distributed data lumping, describing by equations 6.25 to 6.39, is being implemented on three datasets in this subsection. The first dataset corresponds to dataset 3 described previously which contains four static heat sources. The second dataset corresponds to dataset 4 described previously which contains four moving heat sources. Finally, the third dataset was newly generated with heat sources randomly appearing and disappearing within each core.

Table 6.2 shows the comparison of results for the unoptimized and optimized case using the dataset with static heat sources. Column 1 gives the path configuration. The first configuration had a critical path of 12 nodes while the second one had 15 nodes. Columns 2 to 5 contain information pertaining to the unoptimized case while

Path	Unoptimized				Optimized				
Conf.	BW	% Loss	Avg Delay	Avg Error	Opt Ratio	%Loss	%Lump Loss	Avg Delay	Avg Error
1	2	3	4	5	6	7	8	9	10
Path 1	BW 1	43.75	18.65	7.94	Opt 1	60.69	27.16	23.44	6.62
	BW 2	56.25	22.22	7.03	Opt 2	64.7	37.64	20.45	6.38
	BW 3	63.54	30.62	7.17	Opt 3	58.85	52.6	19.93	4.93
Path 2	BW 1	43.75	17.78	5.65	Opt 1	63.58	16.3	23.68	5.21
	BW 2	56.25	22.45	5.92	Opt 2	67.29	43.39	31.28	5.56
	BW 3	63.54	30.82	5.46	Opt 3	71.05	60.52	22.67	4.45

Table 6.3: Comparison of results: Dataset with moving heat sources

columns 6 to 10 gives results for the optimized scenario.

Column 2 mentions 3 values of bandwidth that were used for conducting experiments, with *BW 1* being the fastest and *BW 3* as the slowest. The *'% Loss'* indicates the percentage of packets that were lost during transmission and hence did not reach the target point. Since the bandwidth and buffer size are constant, the *% loss* is the same for both paths in the unoptimized case. Column 4 gives the average delay per packet (that was received at the target point) while column 5 gives the average lumping error per packet (that was lost). In this case, it can be noticed that the as the BW decreases from fastest to slowest, the *% loss* as well as the average delay increases for both path configurations. This is because the packets travel slowly towards the target point and the probability of data being overwritten in the buffer of forwarding nodes is higher. The average lumping error however fluctuates with no preference to specific BW because the unoptimized procedure assigns the same importance to all packets resulting in the packets getting dropped randomly.

For the optimized case, experiments were performed using 3 values of the optimization ratio given by 'error/latency' (ζ/μ) in the cost function given by equation 6.35. The ratio *Opt 1* is 0.7/0.3, *Opt 2* is 0.5/0.5 and *Opt 3* is 0.3/0.7. Columns 7, 9 and 10 for the optimized case are analogous to columns 3, 4 and 5 of the unoptimized case. Column 8 gives the percentage of packets that were dropped due to lumping of nodes. From the results, it can be observed that the the *% loss* for optimized case is mostly greater than the loss for unoptimized case. This is expected since some packets are intentionally dropped by the lumping procedure which is in addition to the loss owing

Path	Unoptimized				Optimized				
Conf.	BW	% Loss	Avg Delay	Avg Error	Opt Ratio	%Loss	%Lump Loss	Avg Delay	Avg Error
1	2	3	4	5	6	7	8	9	10
Path 1	BW 1	43.75	18.65	5.43	Opt 1	63.06	23.86	18.90	4.37
	BW 2	56.25	22.22	4.8	Opt 2	66.04	36.41	23.52	4.79
	BW 3	63.54	30.62	4.88	Opt 3	59.44	51.66	21.47	4.02
Path 2	BW 1	43.75	17.78	4.04	Opt 1	64.13	27.71	24.09	3.05
	BW 2	56.25	22.45	4	Opt 2	70.37	48.76	29.31	3.41
	BW 3	63.54	30.82	3.71	Opt 3	70.43	62.9	23.8	2.76

Table 6.4: Comparison of results: Dataset with random heat sources

to limited buffer size. From column 8 it can be seen that a major part of the loss is actually caused by lumping and is therefore beneficial. This is also reflected in the values of average error which are consistently smaller for the optimized case.

The ratio *Opt 3* puts more emphasis on reducing latency while *Opt 1* focuses more on reducing average error. So, for *Opt 1*, less nodes are lumped and preference is given to lower bandwidths while for *Opt 3*, more nodes are lumped and faster bandwidths are used since we try to reduce latency. Intuitively, the delay for *Opt3* should be least while error for *Opt1* should be lowest. However, in actuality, this may not always be the case since buffer size plays an important role in deciding which packets reach the target point. For example, the less lumping and lower bandwidths of *Opt 1* cause increase in congestion on the paths resulting in loss of important packets.

The results for the dataset with moving heat sources are as shown in table 6.3. For the unoptimized case, the *% loss* and average delay increases for slower bandwidths for the same reason as mentioned before. Also, the average error varies in a random fashion but is obviously larger than the error for the optimized case. For the optimized case, especially for *Opt 3*, it can be seen that the delay is very less as compared to the unoptimized case. Also, the contribution of the lumping procedure towards the total loss of packets is high. Therefore, the average error is low.

The results for the random dataset are mentioned in table 6.3. Similar to the previous datasets, for the unoptimized case, the *% loss* and average delay increases for slower bandwidths and the average error is higher as compared to the optimized case. An interesting observation is that the values of average error are the lowest for

the random dataset as compared to the previous two datasets.

6.7 Conclusion

This chapter provides a method to produce data models from raw sampled data using differential equations given by equation 6.3. Using sampled data over time, the model parameters are produced that correspond to *gradient coefficients* to capture the data flow between physical points and *storage coefficients* to model the density of sampled data at a particular physical point. These model parameters are transmitted to the target point along data paths.

The procedure of distributed data lumping is introduced, where these data models are lumped to reduce the load in the network. The process of lumping helps improve latency and data loss in the network, but introduces errors for dropping nodes as data is forwarded along the path. Equations for lumping errors and related error bounds are derived and through experiments, lumping errors are characterized and it was concluded that lumping errors increase in locations of high density sensor readings and situations that result high gradients with respect to neighboring nodes in the path. These situations need to be captured by the modeling process to enable efficient decision making at the target point. Equations 6.25 to 6.35 are formulated to select the right lumping hierarchy levels and utilization rates between bandwidth values depending on the coefficients of the cost function and experiments correlate data loss to lumping error. Buffer constraints produce data loss at the input buffers of the nodes and important information is lost in the form of errors at the target point. The optimization procedure minimizes average error per lost packet.

Depending on data paths that are used to stream data towards the target point, correlation errors are introduced. The energy that is not captured due to the missing links correspond to correlation errors. This chapter derives equations to compute correlation errors and the bound values can help in deciding the utilization rates of the path. It was experimentally observed that correlation errors can be reduced by using data paths that would establish links with high gradients.

Chapter 7

Conclusions and future work

The primary objective of this thesis was to produce reliable data models using raw sensor data for accurate data representations under tight resource constraints of the execution platform, while satisfying the timing constraints of the application. This objective was achieved using different algorithmic steps, starting with building dynamic adaptation policies for network parameters to satisfy the performance requirements of the application while tracking physical entities that can be quasi-static or dynamic in nature, during the process of generating and streaming data model parameters to the common collection point.

The performance requirements are specified using a declarative, high-level specification notation in Chapter 2 that correspond to timing, precision and resource constraints of the application and Chapter 3 describes the execution platform that defines middleware routines to transfer executable code to individual nodes and run network-level applications on the grid-type sensor network, that is used to run all experiments that are conducted in this thesis.

Chapter 4 describes a performance optimization model for entities having physical properties that are quasi-static in nature. Adaptation policy design switches between design points representing different performance-cost trade offs to address the goals and constraints of the goal-oriented descriptions. Experiments show that the optimization method scales well with network size and improves power consumption by 21% and data loss by 40%, while reducing delay of slower nodes to satisfy the timing constraints of the application.

Chapter 5 adds the aspect of the dynamic properties of the monitored entities to the adaptation policy design discussed in Chapter 4. The correlations between dif-

ferent data paths and trajectory of the moving entities were used to select the right network resource parameters and data paths to satisfy performance requirements of the application and four trajectory prediction algorithms were discussed to capture different facets of the entity dynamics. The performance of the four algorithms were evaluated based on their latency and data loss improvements compared to the other techniques and the conditions under which each algorithm performs best were identified.

Finally, Chapter 6 uses differential equations to produce data model parameters that include *gradient coefficients*, that capture the data flow between physical points and *storage coefficients*, that capture the density of sampled data at different physical points. The process of lumping these data model parameters while transmitting them to the common collection point along data paths introduces lumping and correlation errors, which were characterized in Chapter 6. Using the equations that bound lumping errors and the identified factors that affect them, an algorithm was devised to establish the required trade off between the goals of keeping the lumping errors within the acceptable limit and reducing communication load on the network and experiments were conducted to control the nodes that would experience data loss by adjusting lumping hierarchy and bandwidth rates to minimize errors and delay at the target point.

7.1 Future Work

The research work conducted for this thesis focuses on building high-precision data models using sampled data that models the behavior of the monitored entities to enable decision making. Within the scope of this work, models are considered highly precise when sufficient amount of data is available at the decision making nodes with a delay value that is lower than the threshold value specified by the application. Future work that would employ decision making procedures can use this robust infrastructure to perform goal-oriented control applications in the form of actuation procedures using the data model parameters, and also add a feedback mechanism between the network infrastructure and the decision making module to improve the quality of data models from the goal-oriented perspective. The feedback mechanism can be also be used by the decision making procedures to devise new data paths or alter existing paths to add or reduce redundancy within parts of the network based on the density and

dynamics of the entities in order to adjust errors related to missed data correlations.

Apart from adjusting the quality and quantity of data models, the decision making procedure can use these data model parameters, along with the sensor data, to identify the activities that take place within the environment to aid goal-oriented control procedures. For example, the temperature data, and the S_c and G_r parameter values that are computed for the case study in Chapter 6 can be used to identify the software applications that are running at different processing elements in the chip using causal relations between the nodes in the network. These causal relations can be represented in the form of Directed Acyclic Graphs (DAGs) and using the probability distributions that describes the relations between different nodes in the DAG structures, decisions related to thermal management within the chip [106] and sensor placement strategies [107] can be made.

Bibliography

[1] Subramanian, V., Doboli, A. et al. A Goal-Oriented Programming Framework for Grid Sensor Networks with Reconfigurable Embedded Nodes. In *ACM Transactions in Embedded Computing Systems*, 2011.

[2] Umbarkar, A., Subramanian, V., and Doboli, A. Low-Cost Sound-based Localization using Programmable Mixed-Signal Systems-on-Chip. In *Microelectronics Journal*, 2010.

[3] Wang, M., Subramanian, V., and Doboli, A. Towards a Model and Specification for Visual Programming of Massively Distributed Embedded Systems. In *Sensors and Transducers Journal*, 2009.

[4] Subramanian, V., Umbarkar, A. and Doboli, A. Decentralized detection and tracking of emergent kinetic data for wireless grids of embedded sensors. In *NASA/ESA Conference on Adaptive Hardware and Systems*, 2012.

[5] Subramanian, V., Umbarkar, A. and Doboli, A. Decentralized Event Detection using Distributed Interrupts in Cyber Physical Systems. In *IEEE Systems Conference*, 2012.

[6] V. Subramanian, A. Umbarkar, A. Doboli, Maximizing the Accuracy of Sound Based Tracking via a Low-Cost Network of Reconfigurable Embedded Nodes, *NASA/ESA Conference on Adaptive Hardware and Systems*, 2011.

[7] Umbarkar, A. and Subramanian, V and Doboli, A. Improved Sound-based Localization through a Network of Reconfigurable Mixed-Signal Nodes. *IEEE Workshop, ROSE*, 2010.

[8] Ferent, C., Subramanian, V, Gilberti, M, and Doboli, A. Linear Programming Approach for Performance-Driven Data Aggregation in Networks of Embedded Sensors. *Design, Automation and Test in Europe, DATE*, 2010.

[9] Subramanian, V. and Doboli, A. PNet: A Grid type Sensor Network of Reconfigurable Nodes. In *IEEE International Conference on Distributed Computing Systems Workshops*, 2009.

[10] Subramanian, V. and Doboli, A. Online adaptation policy design for grid sensor networks with reconfigurable nodes. In *Proceedings of Design, Automation and Test in Europe Conference*, 2009.

[11] *lp_solve solver*. http://lpsolve.sourceforge.net/5.5/.

[12] Bailey-Kellogg, C., Zhao, F. and Yip, K. Spatial aggregation: Language and applications. In *Proceedings of AAAI*, 1996.

[13] Basmadjian, D. *The Art of Modeling in Science and Engineering*. Chapman & Hall, 1999.

[14] Basch, J., Guibas, L. and Hershberger, J. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, April 1999.

[15] Levis, P. and Culler, D. Mate: A tiny virtual machine for sensor networks. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[16] Rinat, R. and Smith, S. Modular internet programming with cells. pages, 257–280, 2002.

[17] Zhao, F., Liu, J., Guibas, L. and Reich, J. Collaborative signal and information processing: An information-directed approach. *Proceedings of IEEE*, 91(8):1199–1209, August 2003.

[18] J. Du Improving the Accuracy of Object Tracking in Three Dimensional WSNs Using Bayesian Estimation Methods. *Embedded and Ubiquitous Computing*, December 2010.

[19] Jajamovich et al Collaborative Sensor Networks with Bayesian Multitarget Tracking and Sensor Localization. *Military Communications Conference (MILCOM)*, 2009.

[20] Juan Liu and Reich and Feng Zhao Collaborative In-Network Processing for Target Tracking. *Journal on Applied Signal Processing*, 2002.

[21] Brooks, R., Ramanathan, P. and Sayeed, A. Distributed target classification and tracking in sensor networks. *Proceedings of IEEE*, 91(8):1163–1171, August 2003.

[22] R. Brooks, C. Griffin, and D. Friedlander Self-organized distributed sensor network entity tracking. *International Journal on High Performance Comput. Applicat.*, 2002.

[23] R. Brooks, and S.S. Iyengar Multi-Sensor Fusion: Fundamentals and Applications With Software. *Englewood Cliffs, NJ: Prentice-Hall*, 1998.

[24] Bowen, J. Formal specification and documentation using z: A case study approach. 2003.

[25] Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E. and Culler, D. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation Conference*, 2003.

[26] Zhao, F., Bailey-Kellog, C. and Fromherz, M. Physics-based encapsulation in embedded software for distributed sensing and control applications. *Proceedings of IEEE*, 91(1):40–63, January 2003.

[27] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M., Hellerstein, J., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F. and Shah, M. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proceedings of CIDR Conference*, 2003.

[28] Whitehouse, K., Sharp, C., Brewer, E. and Culler, D. Hood: A neighborhood abstraction for sensor networks. pages, 99–110, 2004.

[29] Welsh, M. and Mainland, G. Programming sensor networks using abstract regions. In *Proceedings of the Symposium on Networked Systems Design and Implementation*, 2004.

[30] Greenstein, B., Kohler, E. and Estrin, D. A sensor network application construction kit (snack). In *Proceedings of SenSys*, 2004.

[31] Hadjieleftheriou, M., Kollios, G., Bakalov, P. and Tsotras, V. Complex spatio-temporal pattern queries. In *Proceedings of the 31st VLDB Conference*, 2005.

[32] Abadi, D., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S. The design of the borealis stream processing engine. In *Proceedings of the Biennial Conference Innovative Data Systems Research*, 2005.

[33] Xing, Y., Zdonik, S. and Hwang, J.-H. Dynamic load distribution in the borealis stream processor. In *Proceedings of the IEEE International Conference on Data Engineering*, 2005.

[34] Liu, Y. and Smith, S. Interaction-based programming with classages. In *Proceedings of OOPSLA'05*, 2005.

[35] Thepayasuwan, N. and Doboli, A. Layout conscious approach and bus architecture synthesis for hardware-software co-design of systems on chip optimized for speed. *IEEE Transactions on VLSI Systems*, 13(5), May 2005.

[36] Gummadi, K. et al. Macro-programming wireless sensor networks using kairos. In *Proc. International Conference on Distributed Computing in Sensor Systems*, 2005.

[37] Cypress Semiconductor Corporation. Psoc mixed signal array. *Document No. PSoC TRM 1.21*, 2005.

[38] Loo, B.-T., Condie, T., Garofalakis, M., Gay, D., Hellerstein, J., Maniatis, P., Ramakrishnan, R., Roscoe, T. and Stoica, I. Declarative networking: Language, execution and optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2006.

[39] Jeffrey, S., Alonso, G., Franklin, M., Hong, W. and Widom, J. Declarative support for sensor data cleaning. pages, 83–100, 2006.

[40] Chu, D., Tavakoli, A., Popa, L. and Hellerstein, J. Entirely declarative sensor network systems. In *Proceedings of the VLDB Conference*, 2006.

[41] Mainland, G., Welsh, M. and Morrisett, G. Flask: A language for data-driven sensor network programs. In *Harvard Technical Report TR-13-06*, 2006.

[42] Liu, Y. and Smith, S. A formal framework for component deployment. In *Proceedings of OOPSLA'06*, 2006.

[43] Mottola, L. and Picco, G. Logical neighborhoods: A programming abstraction for wireless sensor networks. pages, 150–168, 2006.

[44] He, T., Krishnamurthy, S., Luo, L., Yan, T., Gu, L., Stoleru, R., Zhou, G., Cao, Q., Vicaire, P., Stankovic, J., Abdelzaher, T., Hui, J. and Krogh, B. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Transactions on Sensor Networks*, 2(1):1–38, February 2006.

[45] Girod, L., Mei, Y., Newton, R., Rost, S. and Thiagarajan, A. The case for a signal-oriented data stream management system. In *Proceedings of the Biennial Conference Innovative Data Systems Research*, 2007.

[46] Hwang, J.-H., Xing, Y., Cetintemel, U. and Zdonik, S. A cooperative, self-configuring high-availability solution for stream processing. In *Proceedings of the IEEE International Conference on Data Engineering*, 2007.

[47] Ahmad, Y. and Cetintemel, U. Declarative temporal data models for sensor-driven query processing. In *Proceedings of the International Workshop on Data Management for Sensor Networks*, 2007.

[48] Chu, D., Popa, L., Tavakoli, A., Hellerstein, J., Levis, P., Shenker, S. and Stoica, I. The design and implementation of a deductive sensor network system. In *Proceedings International Conference on Embedded Networked Sensor Systems*, 2007.

[49] Sun, P., Wei, Y. and Doboli, A. Flexibility-oriented design methodology for reconfigurable delta sigma modulators. In *Proceedings of Design, Automation and Test in Europe Conference*, 2007.

[50] Soma, R. et al. A semantic framework for integrated asset management in smart oilfields. pages, 119–126, 2007.

[51] Mainland, G., Morrisett, G., Welsh, M. and Newton, R. Sensor network programming with flask. pages, 385–386, 2007.

[52] Tatbul, N., Cetintemel, U. and Zdonik, S. Staying fit: Efficient load shedding techniques for distributed stream processing. In *Proceedings of VLDB Conference*, 2007.

[53] Mueller, R., Alonso, G. and Kossmann, D. Swissqm: Next generation data processing in sensor networks. In *Biennial Conference on Innovative Data Systems Research*, 2007.

[54] Jiang, Y.-C. and Wang, J.-F. Temporal partitioning data flow graphs for dynamically reconfigurable computing. *IEEE Transactions on VLSI*, 15(12):1351 – 1361, Dec. 2007.

[55] Dearle, A., Balasubramaniam, D., Lewis, J. and Morrison, R. A component-based model and language for wireless sensor network applications. pages, 1303–1308, 2008.

[56] Lee, E. Cyber physical systems: Design challenges. In *University of California, Berkeley Technical Report No. UCB/EECS-2008-8*, 2008.

[57] Sun, P., Gilberti, M., Doboli, A., Curiac, D. and Pescaru, D. Dynamic reconfiguration of mixed-domain embedded systems for applications with variable performance requirements. In *Proceedings of Adaptive Hardware and Systems*, 2008.

[58] Werner-Allen, G., Dawson-Haggerty, S. and Welsh, M. Lance: Optimizing high-resolution signal collection in wireless sensor networks. In *Proceedings of SenSys*, 2008.

[59] Hnat, T., Sookoor, T., Hooimeijer P., Weimer W. and Whitehouse, K. Macrolab: A vector-based macroprogramming framework for cyber-physical systems. In *Proceedings of SenSys*, 2008.

[60] Lorincz, K., Chen, B.-R., Waterman, J., Werner-Allen, G. and Welsh, M. Resource aware programming in the pixie os. In *Proceedings of SenSys*, 2008.

[61] Subramanian, V. *A Sensor Network for Environmental Monitoring using PSoCs.* MS Thesis, Stony Brook University, Department of Electrical and Computer Engineering, 2008.

[62] Ahmad, Y., Papaemmanouil, O., Cetintemel, U. and Rogers, J. Simultaneous equation systems for query processing on continuous-time data streams. pages, 666–675, 2008.

[63] Gupta, H., Zhu, X. and Xu, X. Deductive framework for programming sensor network. pages, 281 – 292, 2009.

[64] Umbarkar, A. *Implementation of Phase-based Sound Localization Technique on PSoC*. MS Thesis, Stony Brook University, Department of Electrical and Computer Engineering, 2009.

[65] Munir, A. and Gordon-Ross, A. An mdp-based application oriented optimal policy for wireless sensor node. In *Proceedings of CODES+ISSS*, 2009.

[66] Ferent, C. and Doboli, A. Pnetmap: Virtual network implementation on a partially-known physical network. In *Proceedings of International Workshop on Dependable Network Computing and Mobile Systems*, 2009.

[67] Kallakuri, S. et al. Customization of arbitration policies and buffer space distribution using continuous time markov decision processes. *IEEE Trans. VLSI*, 15(2), 2007.

[68] Murali, S. et al. A methodology for mapping multiple use-cases onto networks on chips. *Proc. DATE*, pp. 118–123, 2006.

[69] Chong, C.-Y. et al. Sensor networks: Evolution, opportunities, and challenges. *Proc. IEEE*, 91(8):1247–1256, 2003.

[70] Benini, L. and De Micheli, G. Networks on chips: A new soc paradigm. *IEEE Computer*, 35(1):70–78, 2002.

[71] Bertozzi, D. et al. Noc synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. Parallel and Distributed Systems*, 16(2):113–129, 2005.

[72] Dally, W. and Towles, B. Route packets, not wires: On-chip interconnection networks. In *Proc. DAC*, 2001.

[73] Hansson, A. et al. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proc. CODES+ISSS*, 2005.

[74] Ascia, G. and Catania, V. and Palesi, M. Multi-objective mapping for mesh-based noc architectures. *Proc. CODES+ISSS*, pp. 182–187, 2004.

[75] Guz, Z. et al. Efficient link capacity and qos design for network-on-chip. *Proc. DATE*, pp. 9–14, 2006.

[76] Brooks, R. et al. Self-organized Distributed Sensor Network Entity Tracking. *Int. Journal High-Perf. Computer Archit.*, Vol 16. No. 3, 2002.

[77] Halupka, D. et al. Robust Sound Localization in 0.18 um CMOS. *IEEE Trans. Signal Processing*, 53(6), 2005.

[78] Lu, H. et al. Energy-efficient Spatially-adaptive Clustering and Routing in Wireless Sensor Networks. *Proc. DATE*, 2010.

[79] Mainland, G. et al. Decentralized, Adaptive Resource Allocation for Sensor Networks. *Proc. Symposium on Networked System Design and Implementation*, 2005.

[80] Hnat, T. et al. A modular and extensible Macroprogramming Compiler. *Proc. ACM SESENA*, 2010.

[81] J. Han and M. Kamber. *Data Mining.* Morgan Kaufman, 2006.

[82] M. Singh, A. Bakshi, and V. Prasanna, Constructing Topographic Maps in Networked Sensor Systems. *Workshop on AlgorithmS for Wireless and Mobile Networks (ASWAN)*, 2004.

[83] R. Newton, and M. Welsh, Region Streams: Functional Macroprogramming for Sensor Networks, *Proc. First Internat'l Workshop on Data Management for Sensor Networks (DMSN)*, 2004.

[84] B. Krishnamachari, and S. Iyengar, Efficient and Fault-tolerant Feature Extraction in Sensor Networks, *2nd Workshop on Information Processing in Sensor Networks*, 2003.

[85] M. Singh, and V. Prasanna, A Hierarchical Model for Distributed Collaborative Computation in Wireless Sensor Networks, *International Journal of Foundations of Computer Science (IJFCS)*, Vol 15(3), 2004.

[86] L. Guibas, Kinetic Data Structures, *Handbook of Data Structures and Applications*, 2004.

[87] R. Sugihara and R. Gupta, Programming models for sensor networks: A survey, *ACM Transactions on Sensor Networks*, 4(2), March 2008.

[88] A. Czarlinska, D. Kundur. Reliable Event-Detection in Wireless Visual Sensor Networks Through Scalar Collaboration and Game-Theoretic Consideration. *IEEE Transactions on Multimedia, Vol. 10, No. 5*, August 2008.

[89] Y. Zhang, N. Meratnia, and P. Havinga. Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *Proc. IEEE Communications Surveys and Tutorials, Vol. 12, No. 2*, 2010.

[90] G. Wittenburg et al. A System for Distributed Event Detection in Wireless Sensor Networks. *IPSN*, April 2010.

[91] T. Banerjee, K. Chowdhury, and D. Agrawal. Distributed data aggregation in sensor networks by regression based compression. In *Proceedings of MASS*, 2005.

[92] P. Bonissone, R. Subbu, and J. Lizzi. Multicriteria decision making (mcdm): A framework for research and applications. *IEEE Computational Intelligence*, 4(3):48–61, 2009.

[93] D. Curiac, C. Volosencu, D. Pescaru, and A. Doboli. Using wireless sensor-controller networks for distributed control in high reliability applications. In *IEEE International Conference on Computer Communication and Networks (ICCCN), Sensor Networks Workshop*, 2008.

[94] M. Fauvel, J. Chanussot, and J. Benediktsson. Decision fusion for the classification of urban remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 44(10):2828–2838, 2006.

[95] T. He, B. Blum, J. Stankovic, and T. Abdelzaher. Aida: Addaptive application-independent data aggregation in wireless sensor networks. *ACM Transactions on Embedded Computing Systems*, 3(2):426–457, May 2004.

[96] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of MobiCOM*, pages 174–185, 1999.

[97] C. Intanagonwiwat, D. Estrin, H. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of International Conference on Distributed Computing Systems*, 2002.

[98] J.-M. LeCaillec and R. Garello. Nonlinear system identification using autoregressive quadratic models. *Signal Processing*, 81(2):357–379, 2001.

[99] C. Lu, B. Blum, T. Abdelzaher, J. Stankovic, and T. He. Rap: A real-time communication architecture for large-scale wireless sensor networks. In *Proceedings of IEEE RTAS*, 2002.

[100] S. Madden, M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *Proceedings of ACM Symposium on Operating System Design and Implementation*, 2002.

[101] S. Maeng, P. Guha, F. Udrea, A. Syed, S. Santra, J. Gardner, J. Park, S.-H. Kim, S. Moon, J.-D. Kim, Y. Choi, and W. Milne. Soi cmos-based smart gas sensor systems for ubiquitos sensor networks. *ETRI Journal*, 30(4):516–525, August 2008.

[102] P. Wang, C. Li, and J. Zheng. Distributed data aggregation using clustered slepian-wolf coding in wireless sensor networks. In *Proceedings of International Conference on Communications*, pages 3616–3622, 2007.

[103] F. Xue, R. Subbu, and P. Bonissone. Locally weighted fusion of multiple predictive models. In *Proc. of International Joint Conference on Neural Networks*, pages 2137–2143, 2006.

[104] M. Zhu, S. Ding, R. Brooks, Q. Wu, N. Rao, and S. Iyengar. Fusion of threshold rules for target detection in sensor networks. *ACM Transactions on Sensor Networks*, 6(2), 2010.

[105] A. Sridhar et al. 3D-ICE: Fast compact transient thermal modeling for 3D-ICs with inter-tier liquid cooling *ICCAD*, 2010.

[106] Zanini et al. Hierarchical Thermal Management Policy for High-Performance 3D Systems With Liquid Cooling *IEEE Transactions on emerging and selected topics in circuits and systems*, 2011.

[107] A. Nowroz, R. Cochran, S. Reda. Thermal Monitoring of Real Processors: Techniques for Sensor Allocation and Full Characterization. *Design Automation Conference (DAC)*, 2012.

[108] Coskun et al. Temperature Aware Task Scheduling in MPSoCs *DATE*, 2007.

[109] Skadron et al. Temperature-Aware Microarchitecture: Modeling and Implementation. *ACM Transactions on Architecture and Code Optimization*, 2004.

[110] David, H. et al. Memory Power Management via Dynamic Voltage/Frequency Scaling. *ICAC*, 2011.

[111] Atienza, D., Gupta, R., De Micheli, G, et al. Temperature Control of High-Performance Multi-core Platforms Using Convex Optimization. *DATE*, 2008.

[112] H.Mizunuma et al. Thermal modeling for 3D-ICs with integrated microchannel cooling. *ICCAD*, 2009.

[113] P. Kongetira et al. Niagara: A 32-way multithreaded SPARC processor.

[114] A.Sridhar et al. 3D-ICe , http://esl.epfl.ch/3D-ICE.

Appendix A

Network Execution Platform

This appendix presents the detailed procedure of defining network parameters on the execution support, which is a 36-node grid-type PSoC network. PSoC is a system on chip, which offers an 8-bit microcontroller, flash memory for programs, SRAMmemory for data, and programmable digital and analog cells, which are all integrated on the same silicon chip. PSoCs hardware reconfiguration capabilities are important for improving performance by customizing the architecture to the application needs. PSoC nodes were wired together in a grid and communicate with each other through UART modules.

A.1 Packet structure for communication

The execution of the networked sensor nodes is based on different kinds of packet structures for communication: (A) server command packets, (B) data packets, and (C) event packets. Command packets are sent by the server to the SNs to program the parameters of the routines. The packets implement the following functionality: (i) Definition of DARs and their associated parameters, (ii) definition of events and actuation procedures, and (iii) starting and resetting the network. Defining the data regions and their associated parameters includes specifying the x and y coordinates of the bottom-left and top-right corners of a rectangular DAR, the set of alternative target points, regions paths, path probabilities, regions sensing precision (i.e. bitwidth resolutions and time intervals between successive measurements), and regions aggregation function. Data packets transmit data between SNs and the target point accumulating the data over a monitored region. Event packets communicate

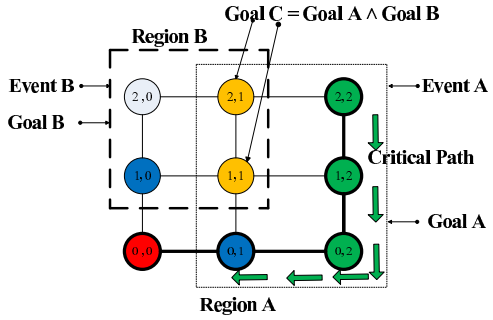

Figure A.1: Middleware Routines for a Region

events over the network. The structure for each of these packets is as follows:

A.1.1 Server Command Packets

Command packets are sent from the server to the PSoC network to define various parameters that update the data structure of the nodes in the network. The nodes perform different functionalities relative to these parameters. The command packets are further divided into the following subtypes:

Define Regions

This command is used to define a set of nodes within the PSoC grid to be a part of the same region. The size of this packet is 7.

Command string format:

$$r \ldots region\_id \ldots x_1 \ldots y_1 \ldots x_2 \ldots y_2 \ldots FFh$$

- r - command id (define region)
- region_id - region name
- x_1, y_1 - coordinates for bottom-left corner of region
- x_2, y_2 - coordinates for top-right corner of region
- FFh - End of Packet

ex. r A 0 1 2 2 *// defines a region named "A", node (0,1) and (2,2) set the boundaries of region "A", node (0,1) being the bottom-left corner of the region and node (2,2) being the top-right corner of the region as shown in Figure A.1.*

Define Region's Target Point

This command establishes which of the nodes associated with a region is the target node. The target node is the node where all the paths within the region will end. The Target node collects data from all the nodes in the network and forms a data pool. The data from the region is fed back to the *Entry Point* and from the *Entry Point* to the server from the target node. The size of this packet is 5.

Command string format:

t...region_id...x...y...FFh

- t - command id (define target point)
- region_id - region name
- x, y - coordinates of target point
- FFh - End of Packet

ex. t A 0 1 *// defines the node (0,1) to be the Target Node for region "A" as shown in Figure A.1 where the Target Point is highlighted in blue.*

Define Region's Path

This command sets a path inside a predefined region. Only one path can be set at a time but a region can have multiple paths. The size of this packet is not fixed as the length of a path is not fixed.

Command string format:

p...region_id...path_id...no_of_nodes...list_of_nodes...FFh

- p - command id (define path)
- region_id - region name
- path_id - path name
- no_of_nodes - total number of nodes on path
- list_of_nodes - x_i, y_i - coordinates of nodes on path, separated by spaces and last node is always the target point

- FFh - End of Packet

ex. p A P 4 2 2 1 2 0 2 0 1 *// defines a path named "P" in region named "A" having 4 nodes with the co-ordinates (2,2), (1,2), (0,2) & (0,1). (0,1) is the target node and the last node in the path. The path is shown in Figure A.1 highlighted in green.*

Define Path Probability

This command is used to set the probability with which a path is chosen within a predefined region. The size of this packet is 5.

Command string format:

q. . . region_id. . . path_id. . . path_probability. . . FFh

- q - command id (define path probability)
- region_id - region name
- path_id - path name
- path_probability - the probability with which a path is chosen (given in %)
- FFh - End of Packet

ex. q A P 30 *// sets the probability of path named "P" in region named "A" to 30% = 0.3.*

Define Region's Precision

This command sets the precision of the data acquisition within a predefined region by specifying the resolution of the ADC (this changes the sampling time) and the time interval between two measurements. All nodes within the region will use these settings. The size of this packet is 5.

Command string format:

s. . . region_id. . . no_of_bits. . . no_of_seconds. . . FFh

- s - command id (define space and time precision within a region)

- region_id - region name
- no_of_bits - ADC bit resolution
- no_of_seconds - time interval in which at least one measurement must be made
- FFh - End of Packet

ex. s A 8 5 *// precision in region named "A" is set to 8 bit ADC resolution and an interval of 5 seconds between 2 measurements.*

Define Region's Event

This command is used to define an event for a region. The size of this packet is 4.

Command string format:

h... region_id... threshold... FFh

- h - command id (define a region's event)
- region_id - region name
- threshold - threshold temperature value that generates an event
- FFh - End of Packet

ex. h A 30 *// defines an event for region "A", 30° Celsius being the threshold temperature value that generates an event.*

Define a Node's Event for a Region

This command is used to define an event for a node (x, y) specific to region "A". The default threshold value for the node for region "A" is the one defined by the command *Define a region's event.* The size of this packet is 6.

Command string format:

n... region_id... x... y... threshold... FFh

- n - command id (define an event for a node)
- region_id - region name

- x, y - coordinates of node
- threshold - threshold temperature value that generates an event
- FFh - End of Packet

ex. n A 3 12 30 *// defines an event for node (3,12) for region "A", 30º Celsius being the threshold temperature value that generates an event. The node may be a part of other regions and they may have different threshold values specific to those regions.*

Define Region's Range

This command defines a range of temperature values for controlling the speed of the fan for a specific region. The command packet specifies three temperature values T1, T2 and T3. If the temperature value is less than T1, the fan is very slow i.e. the PWM which drives the fan has 25% duty cycle. If the temperature value is between T1 & T2, the speed of the fan is slow (PWM with 50% duty cycle). If the temperature value is between T2 & T3, the speed of the fan is medium (PWM with 75% duty cycle) and the speed is fast (PWM with 100% duty cycle) if the temperature value is greater than T3. The size of this packet is 6.

Command string format:

g... region_id... T1... T2... T3... FFh

- g - command id (define region's range)
- region_id - region name
- T1 - temperature value 1
- T2 - temperature value 2
- T3 - temperature value 3
- FFh - End of Packet

ex. g A 20 25 30 *// defines a range of temperature values for region "A" such that the fan is very slow if the temperature value is less than 20º C, speed of the fan is slow for the temperature range of 20º C - 25º C, speed of the fan is medium for the temperature range of 25º C - 30º C and the speed is fast if the temperature value exceeds 30º C.*

Define a Node's Range for a Region

This command defines a range of temperature values for controlling the speed of the fan for a node (x, y) specific to a region "A". The command packet specifies three temperature values T1, T2 and T3. If the temperature value is less than T1, the speed of the fan is very slow i.e. the PWM which drives the fan has 25% duty cycle. If the temperature value is between T1 & T2, the speed of the fan is slow (PWM with 50% duty cycle). If the temperature value is between T2 & T3, the speed of the fan is medium (PWM with 75% duty cycle) and the speed is fast (PWM with 100% duty cycle) if the temperature value is greater than T3. The default range for the node for region "A" is the one defined by the command *Define region's range.* The size of this packet is 8.

Command string format:

o...region_id...x...y...T1...T2...T3...FFh

- g - command id (define node's range for a region)
- region_id - region name
- x, y - co-ordinates of node
- T1 - temperature value 1
- T2 - temperature value 2
- T3 - temperature value 3
- FFh - End of Packet

ex. g A 3 12 20 25 30 *// defines a range of temperature values for node (3, 12) specific to region "A" such that the speed of the fan is very slow if the temperature value is less than 20º C, speed of the fan is slow for the temperature range of 20º C - 25º C, speed of the fan is medium for the temperature range of 25º C - 30º C and the speed is fast if the temperature value exceeds 30º C. The node may be a part of other regions and they may have different ranges specific to those regions.*

Reset Network

This command resets the entire PSoC Network. It is intended as a software version of a hard reset, enabling the user to remotely reset all the nodes in the network, hence deleting all previous information stored in the data structure (region, target, path, path probability, aggregation function, precision, event and range). This command also stops execution. The size of this packet is 1 and it does not include an *End of Packet*.

Command string format:

$$x$$

This command can be also be executed by pressing the reset button of node (0,0) which is the *Entry Point*.

The actual information is not deleted from memory, rather all the indexes in the data structure are reset to zero, this being equivalent to a total loss of information.

Start execution

This command enables execution for the network. All the parameters which include region and it's parameters, events and ranges are defined prior to enabling execution. After the execution is enabled, the nodes in the network start sensing temperature, execute various functions within the regions, check for events and also produce the actuation signals. The size of this packet is 1 and it does not include *End of Packet*.

Command string format:

$$y$$

After this command is defined, the nodes stop execution only on hardware reset or on software reset using the *Reset Network* command.

A.1.2 Data Packets

The *Data Packet* is defined by a node in the network. The node uses this packet format to transmit data associated to it to the Target Point of the region along one of the paths defined by the server. All other nodes in the path just forward this information packet to the subsequent node in the path. The size of this packet is 8.

Command string format:

D... region_id... path_id... x... y... data... funct... FFh

- D - command id (Communicate data to Target Point)
- region_id - region name
- path_id - path used to send data to Target Point
- x, y - co-ordinates of the node
- data - aggregated data computed by the node
- funct - aggregation function used by the node
- FFh - End of Packet

ex. D A P 2 2 17 g *// An information packet defined by node (2,2) which sends data associated to region "A" to the Target Point using path "P".*

A.1.3 Define Event Packet

This information packet is defined by an individual node in case an event occurs for a defined region. An event occurs when the computed data is higher than the threshold value defined for the node and the event is specific for a region. The node defines this packet to inform the Target Point that an event has occurred. The size of this packet is 5.

Command string format:

v... region_id... x... y... FFh

- v - command id (Inform the Target Point about an event)
- region_id - region name
- x, y - co-ordinates of the node
- FFh - End of Packet

ex. v A 2 2 *// An information packet defined by an individual node which transmits the packet to the Target Point.*

Appendix B

4-D Plots for Modeling Errors

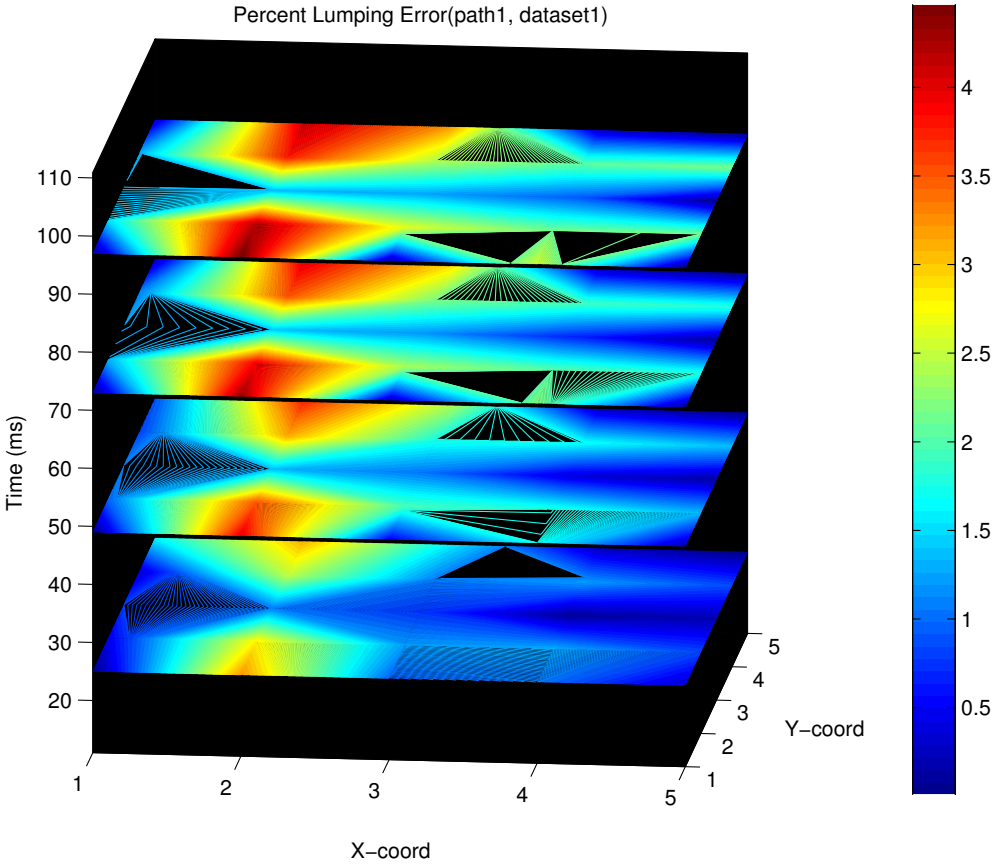

Figure B.1: Percentage lumping errors: path1, dataset1

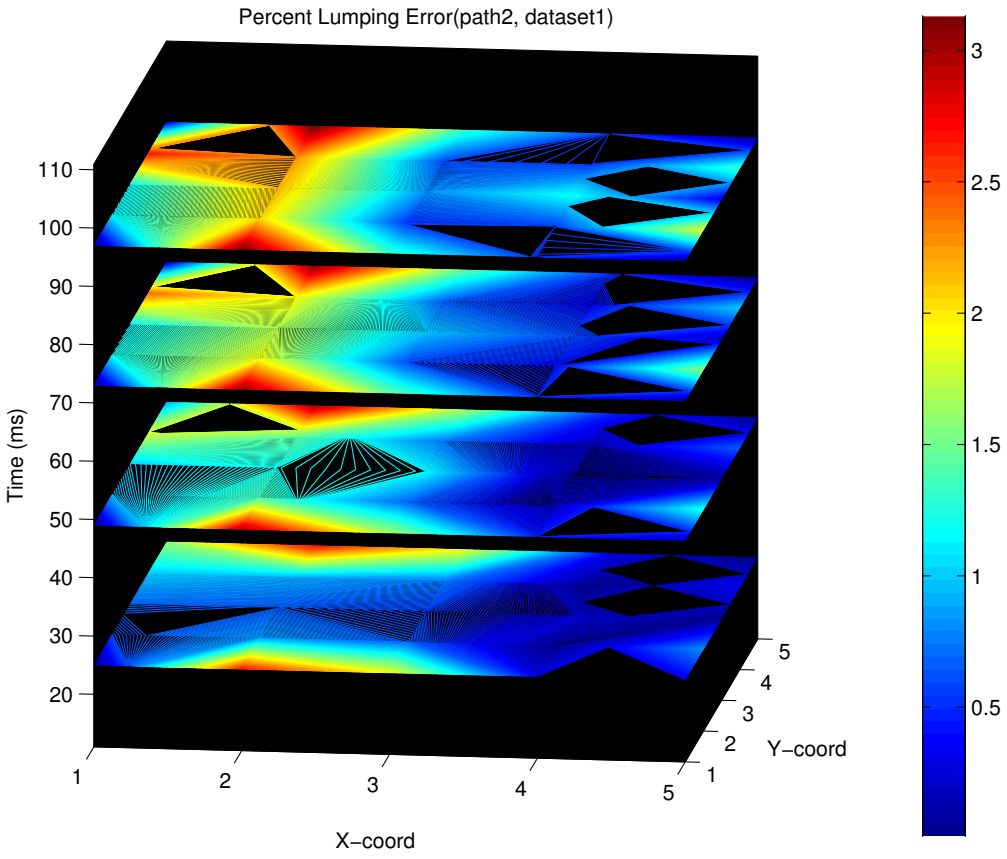

Figure B.2: Percentage lumping errors: path2, dataset1

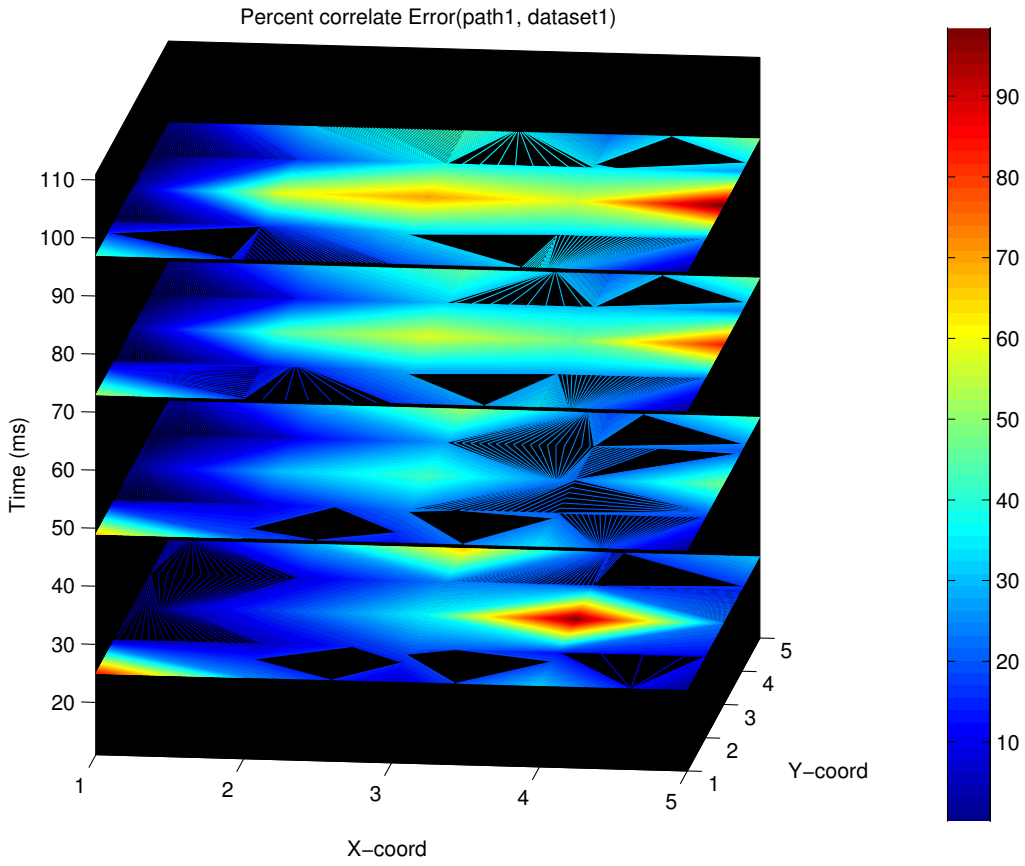

Figure B.3: Percentage correlation errors: path1, dataset1

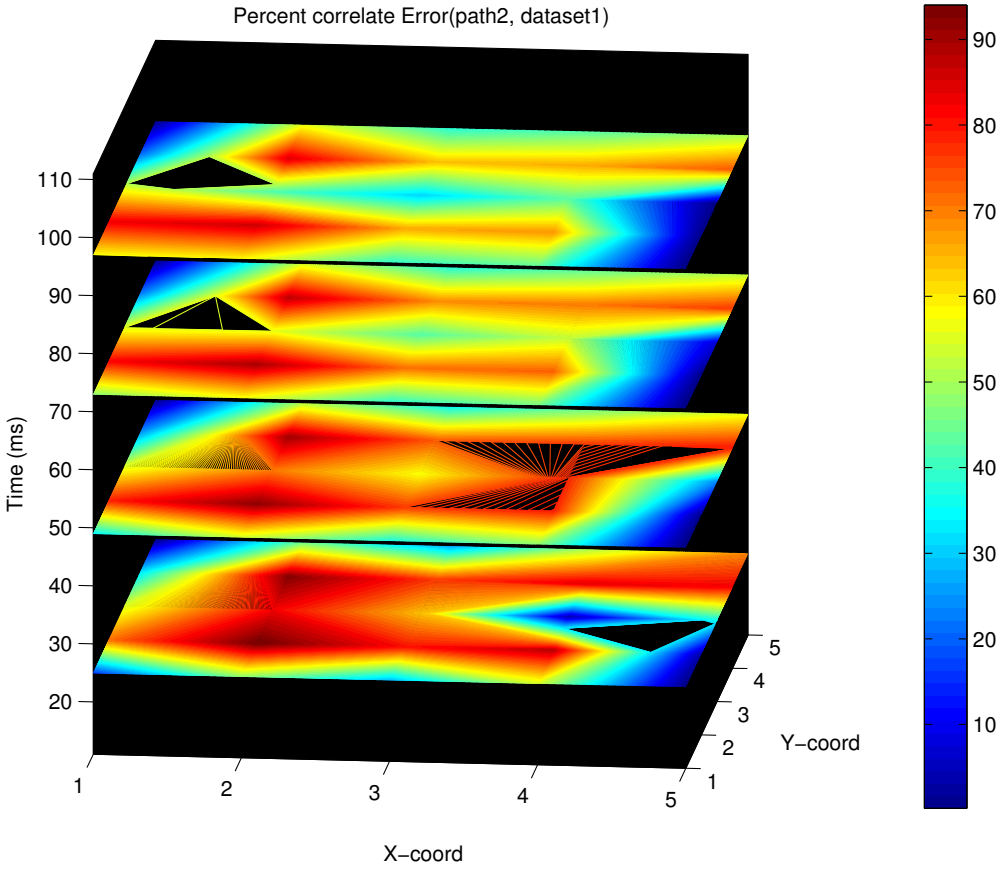

Figure B.4: Percentage correlation errors: path2, dataset1

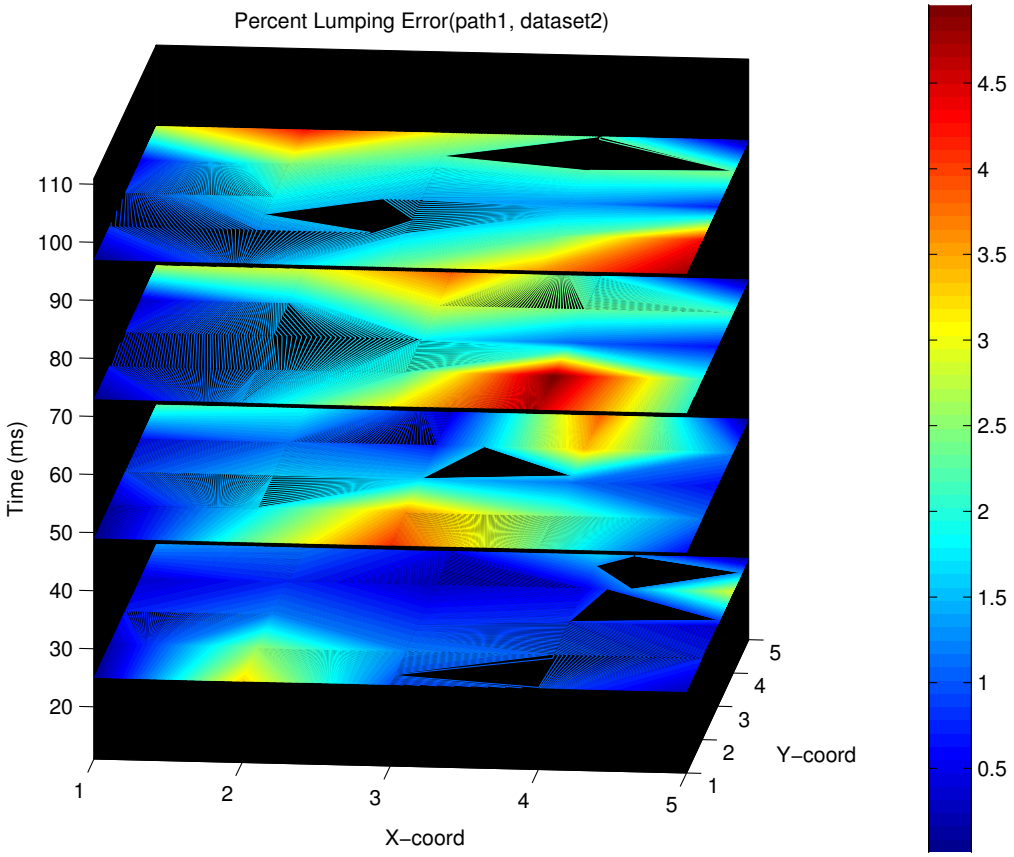

Figure B.5: Percentage lumping errors: path1, dataset2

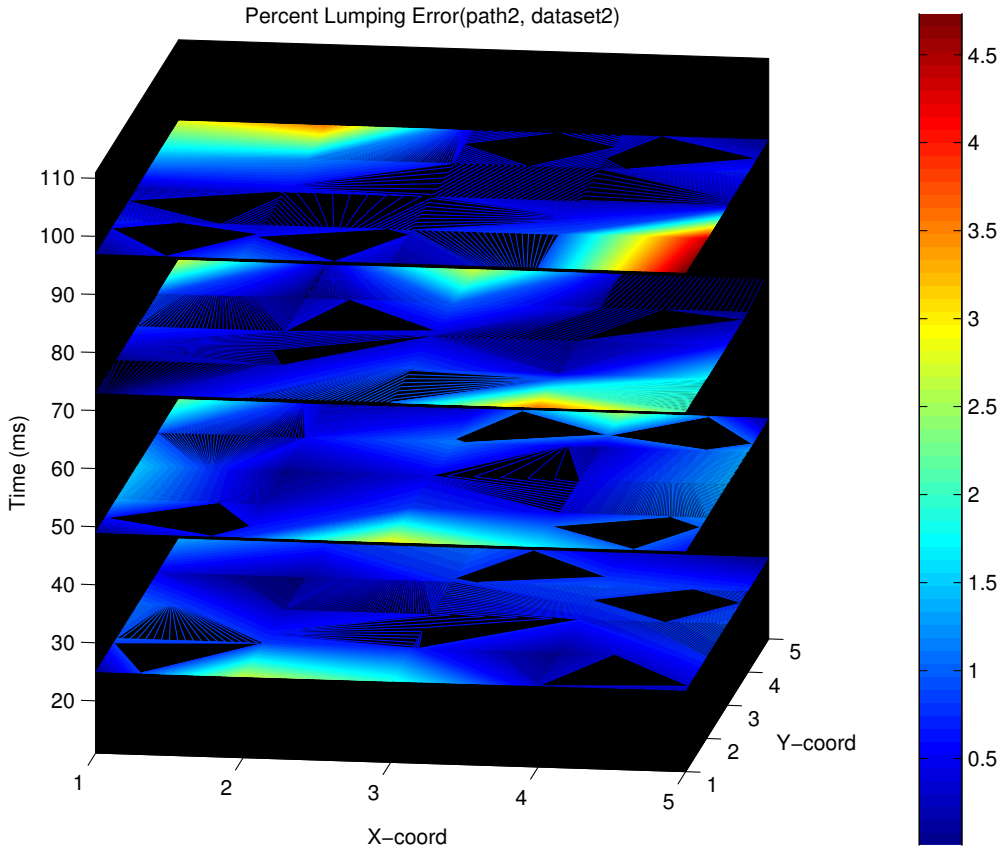

Figure B.6: Percentage lumping errors: path2, dataset2

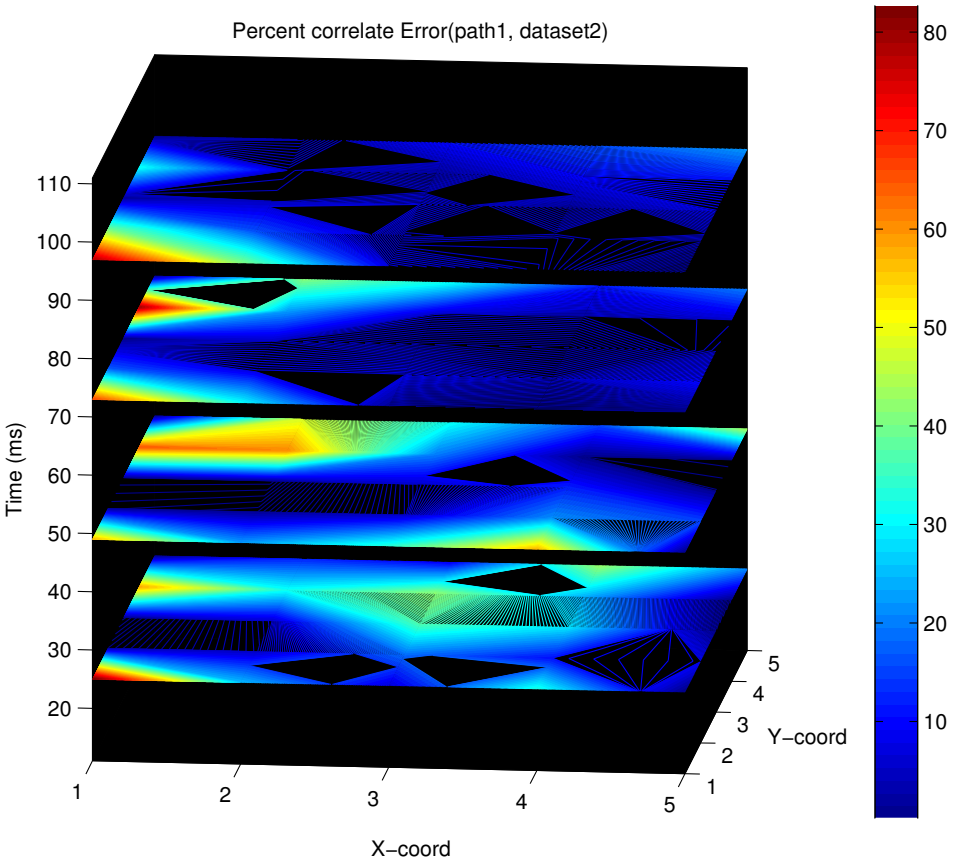

Figure B.7: Percentage correlation errors: path1, dataset2

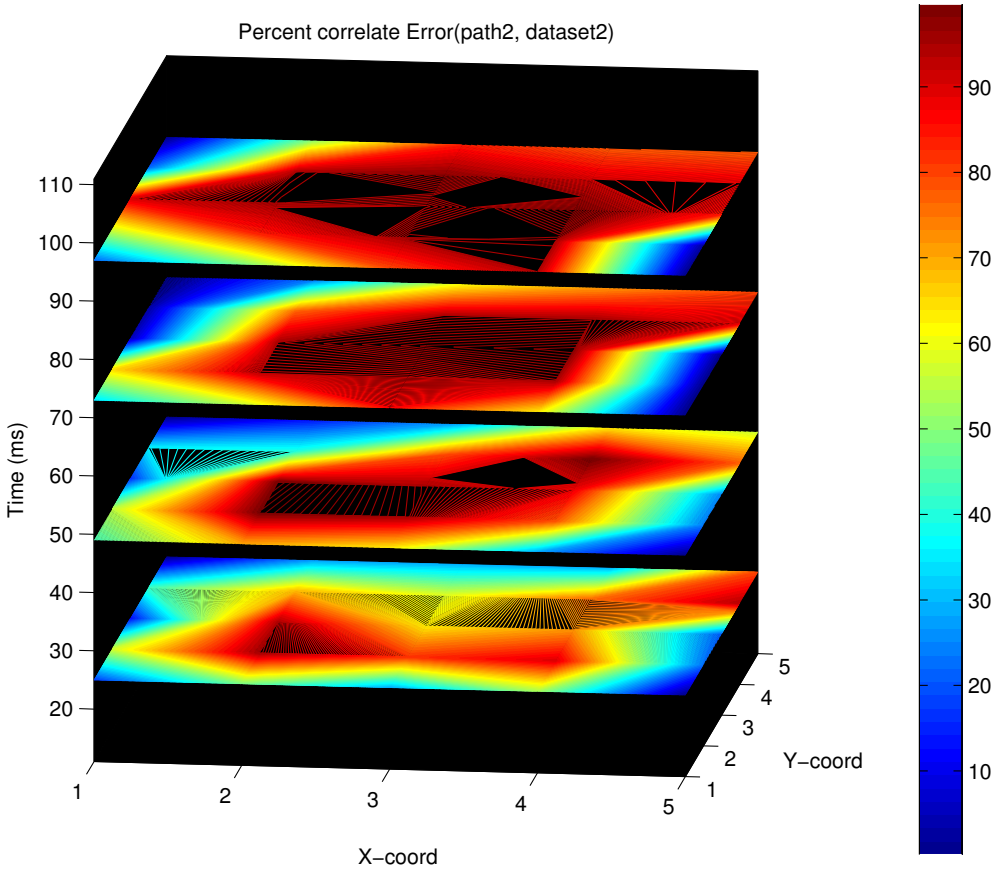

Figure B.8: Percentage correlation errors: path2, dataset2

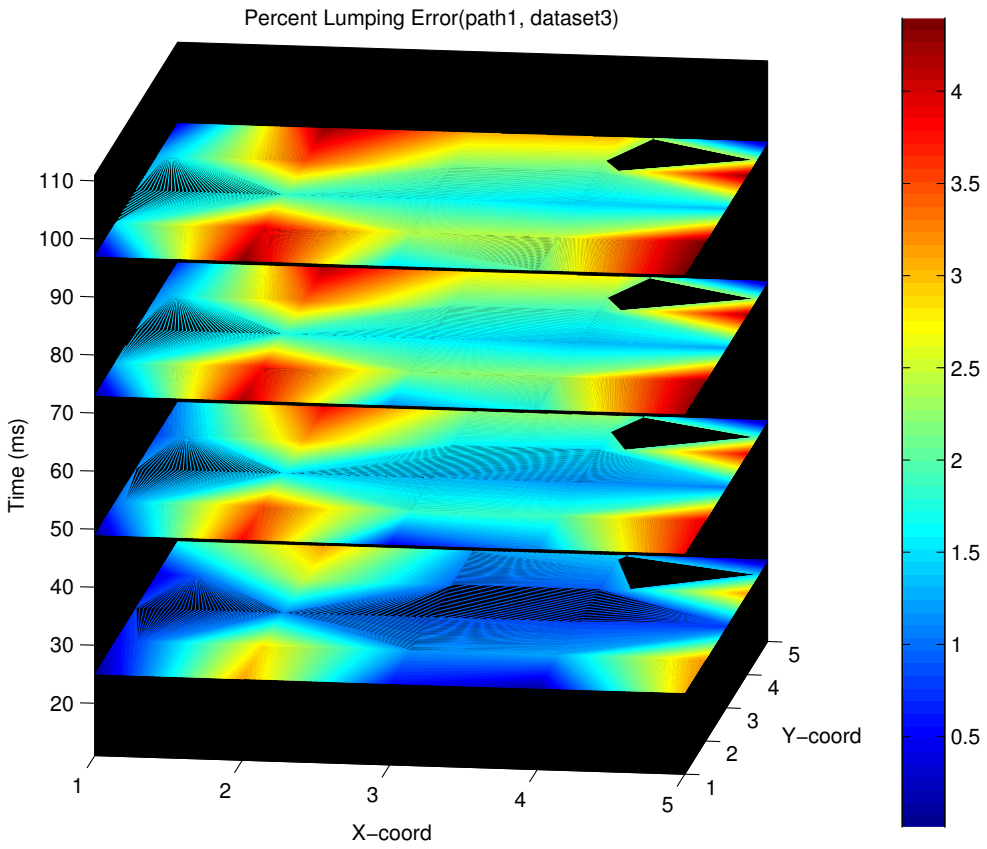

Figure B.9: Percentage lumping errors: path1, dataset3

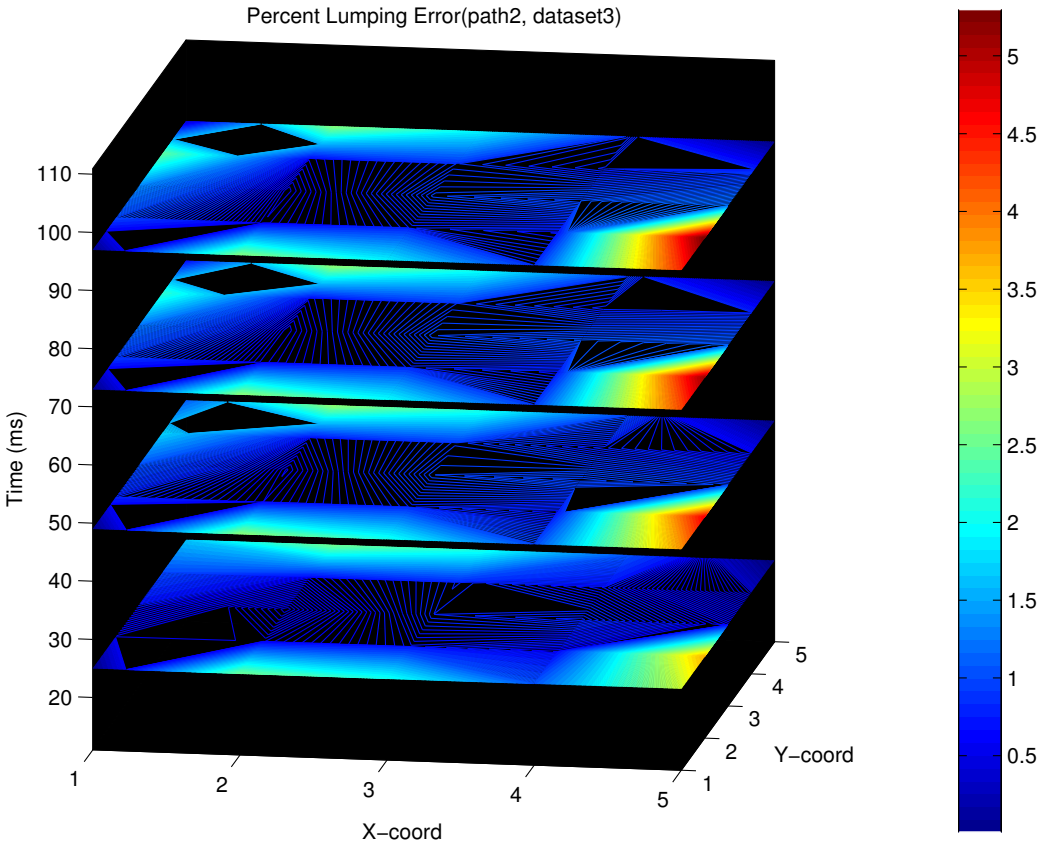

Figure B.10: Percentage lumping errors: path2, dataset3

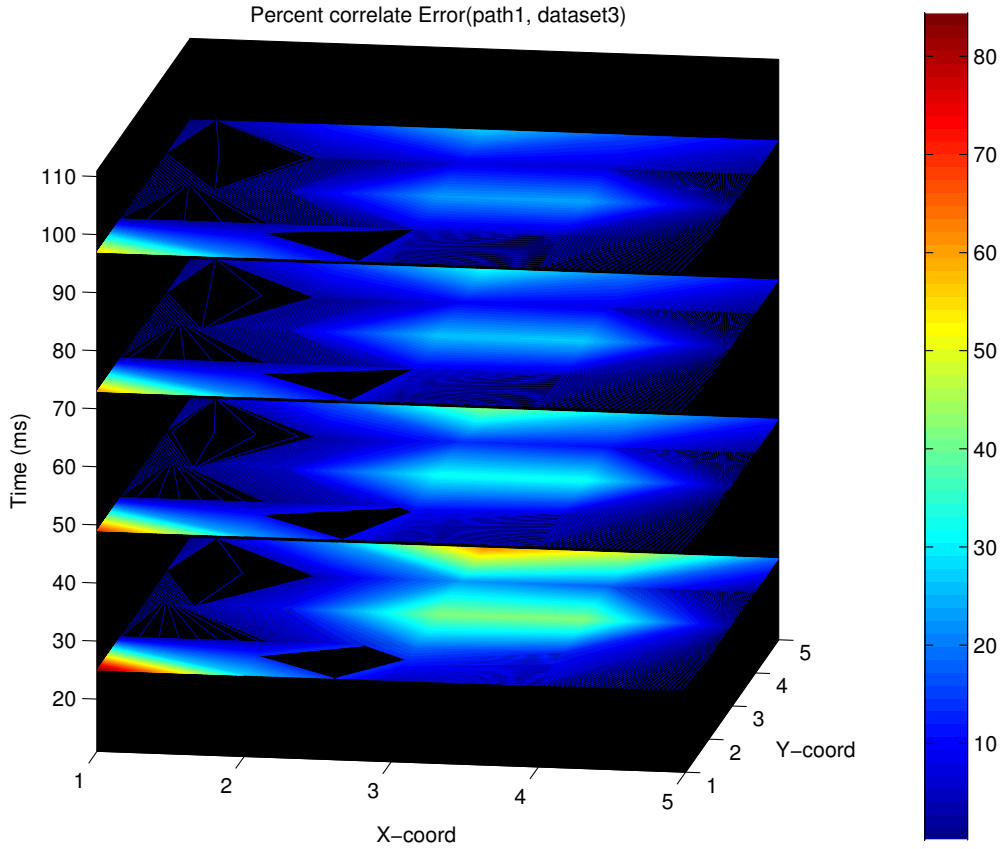

Figure B.11: Percentage correlation errors: path1, dataset3

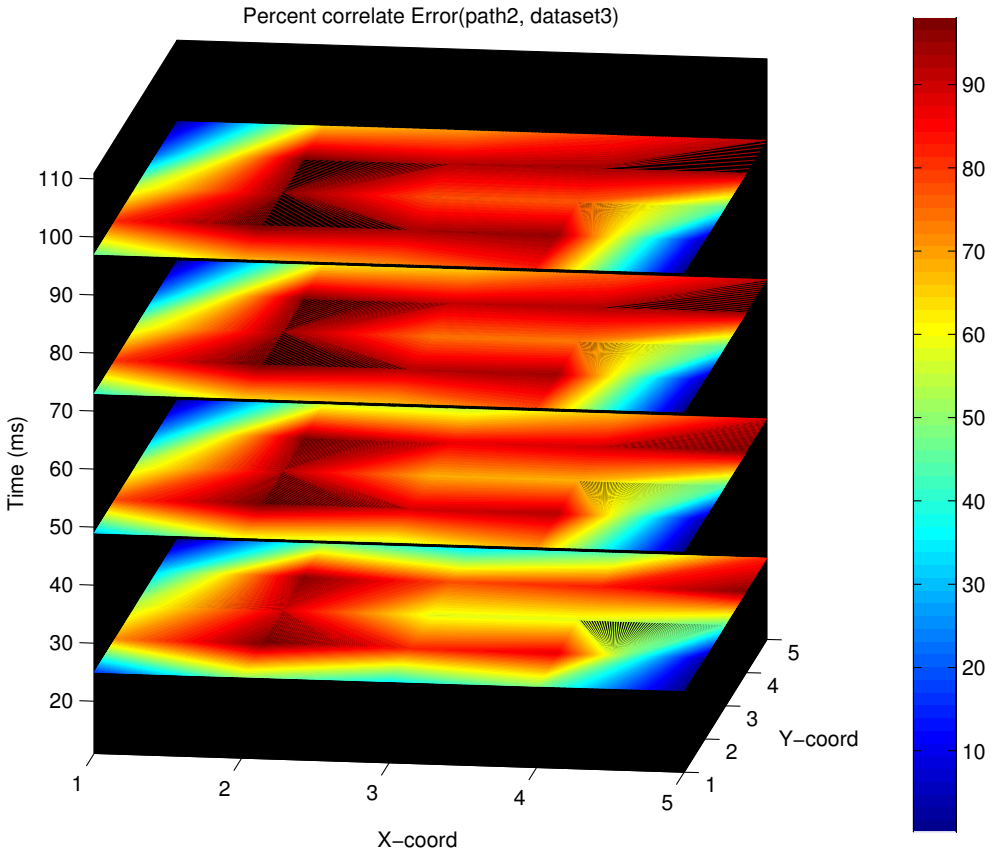

Figure B.12: Percentage correlation errors: path2, dataset3

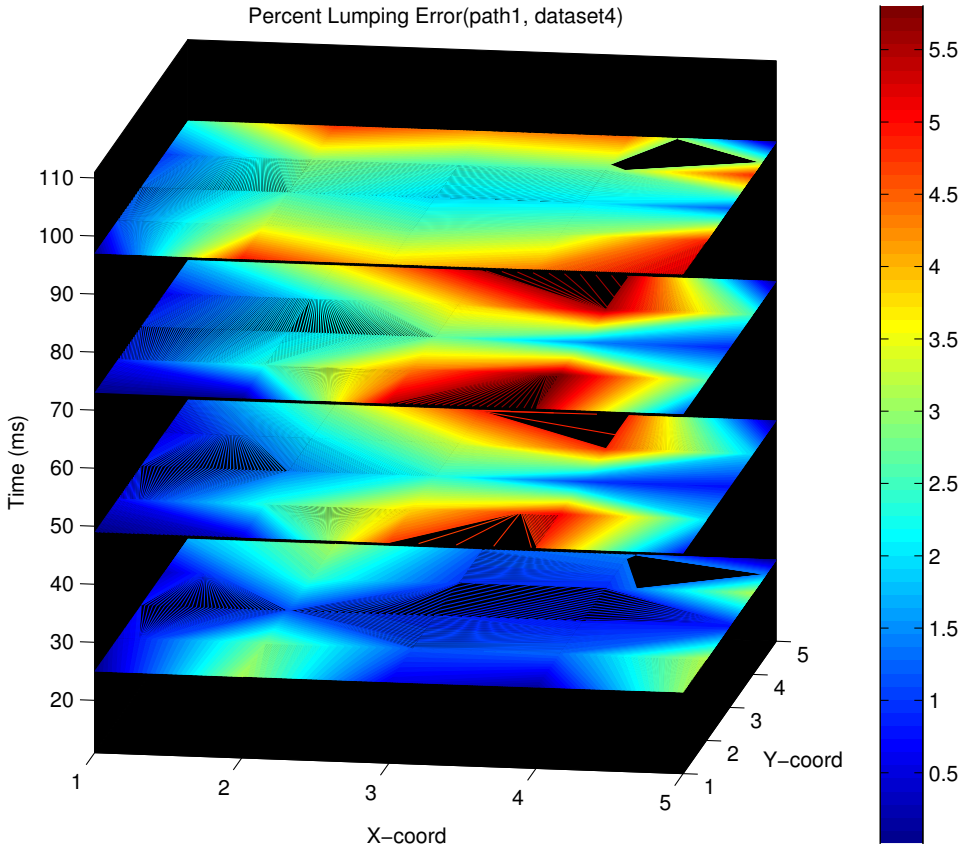

Figure B.13: Percentage lumping errors: path1, dataset4

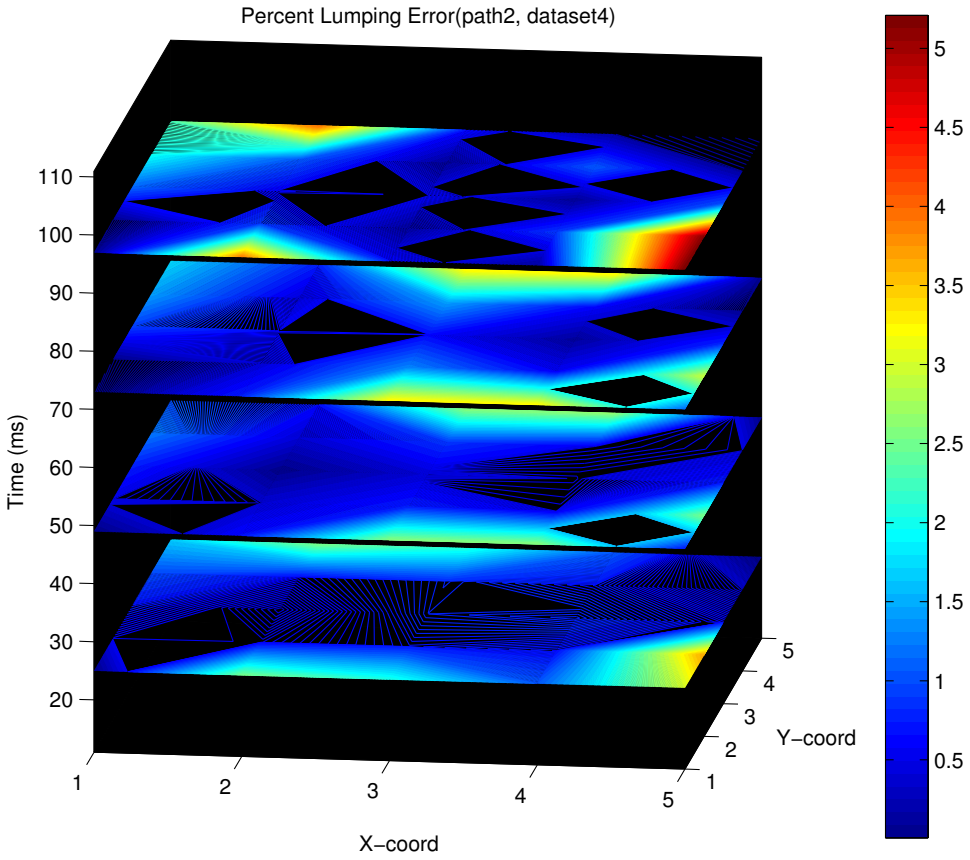

Figure B.14: Percentage lumping errors: path2, dataset4

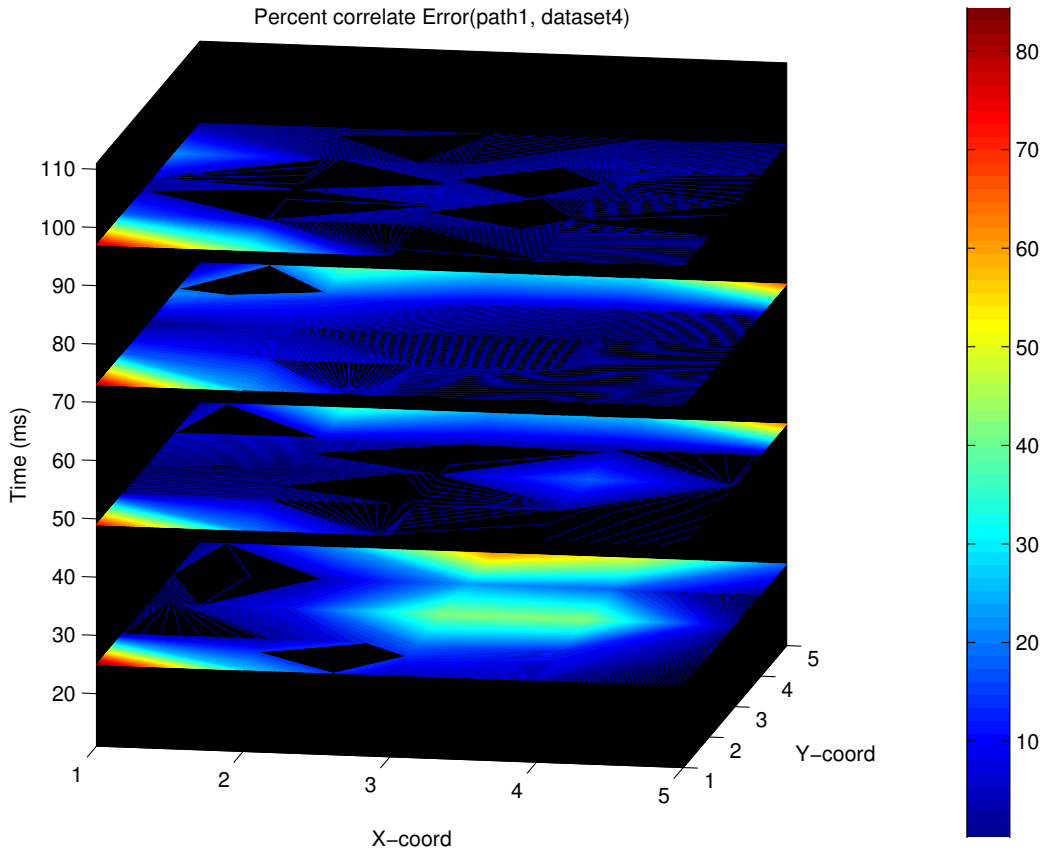

Figure B.15: Percentage correlation errors: path1, dataset4

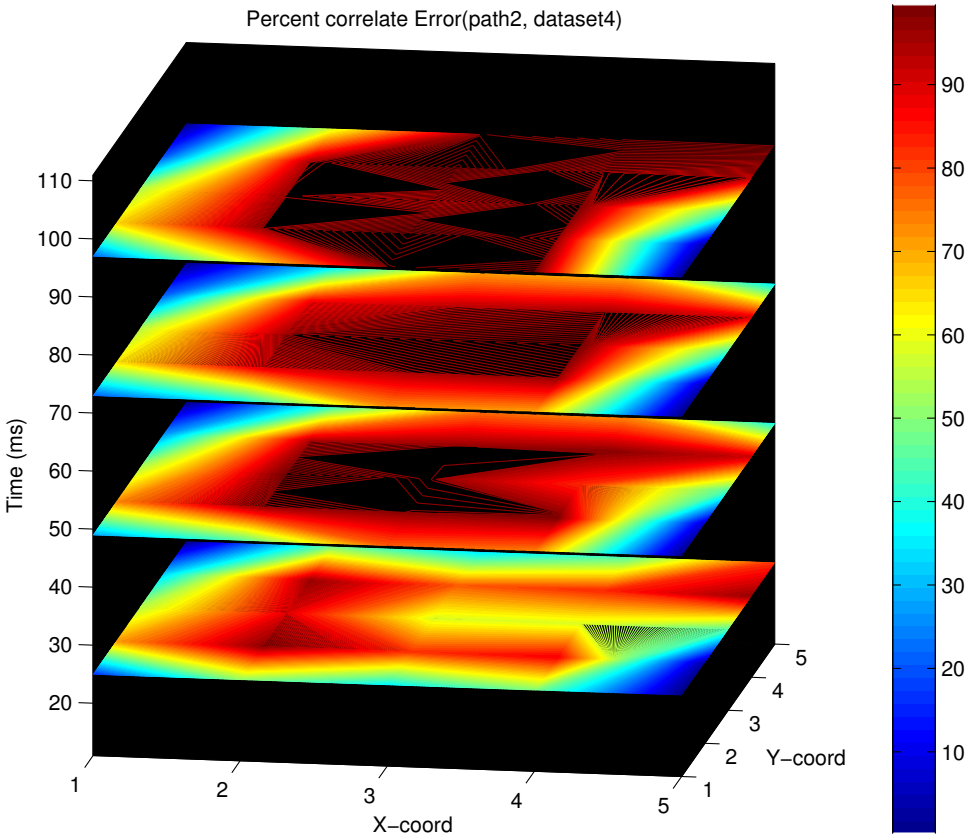

Figure B.16: Percentage correlation errors: path2, dataset4