DISSERTATION

APPLICATION-AWARE IN-NETWORK SERVICE AND DATA FUSION FRAMEWORKS FOR DISTRIBUTED ADAPTIVE SENSING SYSTEMS

Submitted by

Pan Ho Lee

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2009

COLORADO STATE UNIVERSITY

JANUARY 26, 2009

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY PAN HO LEE ENTITLED APPLICATION-AWARE IN-NETWORK SERVICE AND DATA FUSION FRAMEWORKS FOR DISTRIBUTED ADAPTIVE SENSING SYSTEMS BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Committee Member (Dr. V. Chandrasekar)

Committee Member (Dr. Yashwant K. Malaiya)

Committee Member (Dr. Indrajit Ray)

Advisor (Dr. Anura P. Jayasumana)

Department Head (Dr. Anthony A. Maciejewski)

ABSTRACT OF DISSERTATION

APPLICATION-AWARE IN-NETWORK SERVICE AND DATA FUSION FRAMEWORKS FOR DISTRIBUTED ADAPTIVE SENSING SYSTEMS

Distributed Collaborative Adaptive Sensing (DCAS) systems are emerging for applications, such as detection and prediction of hazardous weather using a network of radars. Collaborative Adaptive Sensing of the Atmosphere (CASA) is an example of these emerging DCAS systems. CASA is based on a dense network of weather radars that operate collaboratively to detect tornadoes and other hazardous atmospheric conditions. This dissertation presents an application-aware data transport framework and a multi-sensor data fusion framework to meet the data distribution/processing requirements of such mission-critical sensor applications over best-effort networks. Our application-aware data transport framework consists of a general-purpose overlay architecture and a programming interface. The architecture enables deploying application-aware in-network services in an overlay network to allow applications to best adapt to the network conditions. The application programming interface (API) facilitates development of applications within the architectural framework, and supports the configuration of overlay nodes for the in-network application-aware processing. The API also enables communication between the applications and the overlay routing protocol for the desired QoS support. We demonstrate the efficacy of the proposed framework by considering a DCAS application. In the DCAS application, high-bandwidth radar data is distributed to multiple end users with heterogeneous QoS requirements and different network conditions. We evaluate the proposed schemes in a network emulation environment and on Planetlab, a world-wide Internet test-bed. The proposed

schemes are very effective in delivering high quality data to the multiple end users under various network conditions.

This dissertation also presents the design and implementation of an architectural framework for timely and accurate processing of radar data fusion algorithms in a networked-radar environment. The preliminary version of the framework is used for real-time implementation of a multi-radar data fusion algorithm, the CASA network-based reflectivity retrieval algorithm. As a part of this research, a peer-to-peer collaboration framework for multi-sensor data fusion in resource-rich radar networks is presented. In the multi-sensor data fusion, data needs to be combined in such a manner that the real-time requirement of the sensor application is met. In addition, the desired accuracy of the result from the multi-sensor fusion has to be obtained by selecting a proper set of data from multiple radar sensors. A mechanism for selecting a set of data for data fusion is provided with the consideration application-specific needs. We have also proposed a dynamic peer-selection algorithm, Best Peer Selection (BPS) that selects a set of peers based on their computation and communication capabilities to reduce the execution time required to process data per integration algorithm. Simulation-based results show that BPS can deliver a significant improvement in the execution time for multi-radar data fusion.

As multi-sensor fusion applications have a stringent real-time constraint, estimation of network delay across the sensor networks is important, particularly as they affect the quality of sensor fusion applications. We develop an analytical model for multi-sensor data fusion latency for the Internet-based sensor applications. The ubiquity of time scale-invariant burstiness observed across the network produces excessive network latencies. The multi-sensor data fusion applications require synchronizing a set of correlated data before fusion processing can begin. The analytical model considers the network delay due to the self-similar cross-traffic and latency for data synchronization. A comparison of the analytical model and simulation-based results show that our model provides a good estimation for the multi-sensor data fusion latency. The

model can be used for provisioning of network bandwidth and developing data synchronization strategy for multi-sensor fusion applications.

Pan Ho Lee
Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO 80523
Spring 2009

ACKNOWLEDGMENTS

This thesis would not have been possible without the help, support and guidance of many people. First, I would like to express my special thanks to my adviser, Dr. Anura P. Jayasumana. I have learned a great deal from Dr. Jayasumana, especially to critically and logically approach research problems. Aside from this research, Dr. Jayasumana has continually supported me by offering various resources, and has helped me to broaden my scope of thoughts. I would like to express my gratitude to Dr. V. Chandrasekar for providing me with an opportunity to become a member of CASA. I have thoroughly enjoyed working for CASA. I am also grateful to my other committee members, Dr. Yashwant K. Malaiya and Dr. Indrajit Ray for their valuable comments and suggestions.

It has been my privilege to work with such a talented group of researchers over the past years. I would like to express my special thanks to Dr. Tarun Banka for helping me immensely during the course of my research. Computer Networking Research Laboratory (CNRL) at Colorado State University, especially Dilum Bandara, Saket Doshi and Raghunandan Mandyam Narasiodeyar have provided valuable feedback on my papers and this thesis. I would also like to thank Dr. Sanghun Lim for his help and suggestions during the process of collaborating with him on the data fusion framework development. This work is supported by the Engineering Research Centers Program of the National Science Foundation under NSF award number 0313747.

There are several other people, including Dr. Young Kim and his wife Yeon Hee Lee, Dr. Yohan Yoon, Guihwan Hwang, Sunghwan Woo, and Jaehoon Kim and their families who I would like to thank for their support and friendship throughout my time in Colorado.

My absence has perhaps caused an uneasy time for my family. During this long period of time they have selflessly offered their patience and support, and for this I am greatly appreciative. Finally, I would like to thank my wife Minsook. Her love and support has been the foundation of this work and for all of my other accomplishments.

But seek first his kingdom and his righteousness. Matthew 6:33

DEDICATION

To my parents and parents-in-law

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Following its introduction in the 1980s, the Internet has quickly emerged as a mass market platform for broadband communications by the 1990s. The Internet's influence has rapidly spread across the globe, creating the worldwide network of computers that we are familiar with today. Furthermore, with the advancement of various broadband networking technologies along with high-end personal computers, we are currently expecting much improved video distribution services using IP packets, such as high-definition TV (HDTV) broadcast, interactive TV, and video conferencing over the Internet [Yu04][Ga05][Ki06]. Moreover, it has facilitated technological advances in many fields changing the way we interact with our environment. New types of sensor networks covering large geographical areas are emerging to detect hazardous weather conditions [Cas][Ma04], monitor seismic activity [Cbt] and ecosystems using high-performance sensors such as radars, satellite and radio telescopes. The high-performance sensor nodes that typically run TCP/IP are connected by a combination of wired and wireless networks, and may share communication links with other Internet nodes. Within the high-end sensor networks, the sensor nodes are not resource-constrained in terms of computation and energy compared to mote-based wireless sensor networks, and data generation rates can be several Kbps to tens of Mbps per sensor node. Sensors with low to very high bandwidth requirements, wired

and wireless network infrastructure, and the presence of multiple cross traffic streams in the network poses unique challenges for the data distribution to the diverse end users. Furthermore, the sensor networks are designed as multiple end-user systems. Multiple end-users/applications may have distinct sensing/communication/computation requirements for their operations. Collaborative Adaptive Sensing of the Atmosphere (CASA) [Cas][Mc05], an example of the high-end sensor networks, is a sensor network for atmosphere and weather monitoring using a network of weather radars. In such networks, it is necessary to distribute the captured information from sensors to remote processing nodes or from processing nodes to other remote consumers of the data. The different end-users of the sensor networks typically have their own distinct QoS requirements in terms of delay, throughput, error in end results, and acceptable loss threshold. Furthermore, in these sensor networks, groups of sensors collaborate to sense and detect physical phenomena to provide higher resolution on the information of the hazardous conditions. In these systems, data from different sources may be combined or merged to provide improved accuracies and inferences. Moreover, the real-time sensor network applications require processing of multi-sensor data fusion algorithms within a time bound. Thus, a framework that allows the multi-sensor data fusion algorithms to meet the real-time requirement of the sensor applications is required.

The goal of this dissertation is to provide frameworks to meet the data distribution/processing requirements of such mission-critical sensor applications over best-effort networks. To achieve this goal, the dissertation presents (i) an overlay-based application-aware data transport service framework, and (ii) a multi-sensor data fusion framework.

Using the overlay-based application-aware data dissemination architecture, we seek to provide efficient and flexible system support for application-aware data transport services. We have designed and implemented an Application-aWare Overlay Networks (AWON) architecture, to deploy application-aware services in an overlay network environment in order to best meet the application QoS requirements over the available networking infrastructure. We have also

developed application-aware in-network services to improve the quality of contents delivered to the end-users in bandwidth-constrained conditions using AWON. Effectiveness of the AWON architecture and the application-aware in-network services are demonstrated for a real-time CASA radar data dissemination application.

We present the design and implementation of an architectural framework for data fusion algorithms in a high-end sensor network. The framework offers a data managing and searching mechanism for multi-sensor data fusion applications. It also provides an efficient multi-process coordinating mechanism to distribute computation tasks required by the data fusion algorithms to meet the execution time requirements. The proposed framework is used for implementing a CASA multi-radar data fusion algorithm, the network-based reflectivity retrieval algorithm. Furthermore, a peer-to-peer collaboration framework for multi-sensor data fusion is presented to exploit the parallelism inherent in distributed collaborative sensor networks. We present a data synchronization mechanism and a peer selection strategy to coordinate peers for multi-sensor data fusion applications. Our simulation results illustrate the effectiveness of the proposed framework.

The ubiquity of time scale-invariant burstiness (i.e., self-similarity) across the network produces excessive network latencies. In order to understand how self-similarity of cross-traffic affects the performance of data fusion applications, this dissertation proposes an analytical model for data fusion latency due to self-similar network cross-traffic. Because multi-sensor data fusion applications require the synchronization a set of correlated data item before fusion processing can begin, the analytical model also considers the synchronization delay. A comparison of the analytical model and queuing simulation-based results show that our model provides a good estimation for the multi-sensor data fusion latency.

1.1 Distributed Collaborative Adaptive Sensing Systems

Distributed Collaborative Adaptive Sensing (DCAS) systems are emerging for applications such as seismic activity monitoring [Ctb], distributed target tracking, and weather monitoring [Cas][Mc05]. Collaborative Adaptive Sensing of the Atmosphere (CASA), an example of the DCAS systems, uses a network of low-power X-band radars that detect and predict hazardous weather. CASA is expected to achieve improvements in warning time and forecast accuracy for such weather using distributed, collaborative, adaptive sensing of the lower few kilometers of the atmosphere. Within distributed collaborative adaptive sensing systems such as CASA, the underlying network infrastructure may itself be subjected to adverse conditions due to network congestion, link degradation/outage, and variable cross-traffic along wired and wireless links. Furthermore, high bandwidth data generated by sensors has to be distributed to a heterogeneous set of end-users with different QoS needs, contents requirements and heterogeneous resource availability.

To address these challenges, the network has to be enhanced to provide several network services, such as heterogeneous multicasting and application-aware adaptation. To reduce the load on the communication links, data should be multicast within the network, avoiding multiple identical packets being sent over the same link. Moreover, the heterogeneity in end-user requirements and inherent dynamic resource availability requires an adaptive communication infrastructure. Furthermore, in the DCAS systems, large numbers of distributed sensors operate collaboratively by the coordination of their sensing on the same region in order to achieve higher temporal and spatial resolution [Ku06]. In this scenario, multi-sensor data fusion applications will increasingly become common, and be a crucial part of the systems. Multi-sensor data fusion necessarily involves collecting of data from multiple remote sources and processing computing intensive data fusion algorithms. Thus, new types of network/system support are needed for the

DCAS systems to meet the data processing/distributing requirements, such as application-aware adaptation, multicasting and data fusion.

1.2 Application-Aware Data Dissemination Architecture for DCAS

The basic functionality of the best effort Internet is not enough to meet the desired functionalities and services of the emerging DCAS systems. As seen in the prior Internet QoS proposals, such as IntServ [Br94], DiffServ [Bl98] and IP multicast [Er94], the process of deploying new network services is lengthy and difficult because it requires the modification of the Internet infrastructure. To overcome this problem, many overlay networks were proposed. An overlay network is a network that is built on top of an already existing network to provide a range of useful services for enhancing Internet functionality without the need for changing underlying network infrastructure [An01][Su04]. Some of the overlay networks provide additional services, such as finding better Internet paths, maintaining better multicast tree in the Internet, smoothing packet losses, packet prioritization, and bandwidth guarantees. As previously mentioned, DCAS systems often have to operate under adverse conditions, such as severe weather conditions that may impact part of the network infrastructure. With the use of overlay network architectures, DCAS systems will be able to more effectively adapt to such conditions in the underlying network, and provide the required connectivity and transport quality to applications in a more resilient manner. In addition to the dynamic network conditions, high bandwidth data generated by sensors has to be distributed to a heterogeneous set of end-users with different quality-of service requirements and access bandwidth. To address the unique data distribution challenges in sensor networks, it is required to enable the DCAS applications to adapt to available network resources in an application-specific manner. Some overlay networks allow applications to decide how to address the variation in the network conditions and end-user requirements [Su04]. Thus, overlay-based

solution is an attractive choice to meet the requirements of the DCAS applications. In this dissertation, we explore an overlay based approach in order to offer the required networking services for the mission-critical sensor applications.

1.3 Multi-sensor Data Fusion Framework for DCAS

Within the Wireless Sensor Network (WSN) domain, simple data aggregation techniques, such as maximum, minimum and average of sensor readings are widely used to reduce the overall data traffic to save energy [Ma02]. In high-end sensor networks, such as CASA, multi-sensor data fusion techniques are used to achieve improved accuracies and more specific inferences by combining data from multiple sensors using multi-sensor data fusion algorithms [Lim07]. Data fusion algorithms need to obtain data to be integrated from multiple sensors. Since geographically distributed sensors generate substantial volume of data, the communication cost cannot be overlooked. In addition, large volumes of generated data have to be processed with a stringent time constraint, so vast amount of computation resources have to be deployed on the fly. Client-server is one of the most popular frameworks for realizing multi-sensor data fusion. In a client-server framework, a powerful server acquires data from sensors to perform the algorithms. Although widely used, these frameworks may not be appropriate for multi-sensor data fusion applications, as the demand on the server would increase in proportion to the total number of sensors, quickly overrunning server's limited capacity. To deal with the issue, peer-to-peer (P2P) framework is of interest as an alternative paradigm to the client-server architecture. In the P2P frameworks, the fusion applications can aggregate and utilize unused resources from peers over the network to achieve better performance on processing sensor data. Furthermore, the P2P mechanism also may reduce the risk of a single-point-of-failure which can be disastrous in client-server architecture.

The mission-critical sensor applications have unacceptable performance if long delays are incurred in the network. The ubiquity of time scale-invariant burstiness (i.e., self-similarity) observed in networks introduces complexities into providing QoS for multi-sensor data fusion applications. Therefore, the analysis of network delay across sensor networks due to the bursty cross-traffic is important as it affects the quality of sensor applications. Such analysis facilitates the design of a network infrastructure to satisfy end-to-end delay requirements for mission-critical sensor applications.

1.4 Scope of Dissertation and Objectives

First key contribution of this dissertation is the development of a software framework for deploying in-network services to meet the contents/QoS requirements of the end-users in CASA systems. This dissertation presents an overlay networking-based application-aware transport service framework, which enables applications to access overlay network service and be adaptive to changes in underlying network condition. The framework consists of a general-purpose overlay architecture, called Application-aWare Overlay Network (AWON), and a programming interface to support the deployment of application-aware services on the overlay nodes. We demonstrate the effectiveness of the framework by implementing a CASA application using the architecture and the API. In CASA applications, high-bandwidth radar data is distributed to multiple end-users with heterogeneous QoS requirements. The API is used to implement and to deploy the application-aware services at the source nodes, the intermediate nodes and the multicast nodes of an overlay network. The performance of the proposed framework is evaluated on Planet-lab and additional performance evaluations are executed using a network emulation environment.

The second contribution of this dissertation is to present a design and implementation of an architectural framework for timely and accurate processing of radar data fusion algorithms.

Effectiveness of the proposed framework is demonstrated for a CASA multi-radar data fusion algorithm, the network-based reflectivity retrieval algorithm in CASA-IP1. IP1 is the first test-bed of the Center for Collaborative Adaptive Sensing of the Atmosphere. This dissertation then explores the promise of peer-to-peer computing technologies to offer new possibilities in distributed fusion processing. We present a peer-to-peer collaboration framework for multi-sensor data fusion. In the framework, we propose a data synchronization mechanism and a peer selection heuristic algorithm to coordinate peers for multi-sensor data fusion applications. The data synchronization mechanism selects a set of data for fusion while considering application-specific needs, such as the minimum number of data items from different sensors, the maximum possible waiting time for data when the number of data items is not sufficient, and the maximum tolerance in sampling time difference among the data to be integrated. A dynamic peer-selection heuristic algorithm, Best Peer Selection (BPS), is presented to choose a set of peers based on their computation and communication capabilities to reduce the execution time required for processing data per integration algorithm. We implement a simulation to evaluate the performance of the proposed framework and heuristic algorithm.

Performance modeling is an important part in network design and resource allocation. It allows the prediction of the system performance before installing a real system. This dissertation develops an analytical model for data fusion response time considering network delay due to the self-similar cross-traffic and application-specific requirements. The model calculates the probability that the multi-sensor data fusion latency exceeds a certain time considering network utilization and burstiness. The model can be used for provisioning of network bandwidth and developing data synchronization strategies for multi-sensor fusion applications.

1.5 Dissertation Outline

The remainder of this dissertation is organized as follows:

Chapter 2 revisits the problems presented in this chapter and discusses them in relation to background work. Different QoS provisioning techniques are compared to identify their effectiveness and limitation for the emerging DCAS applications. This chapter also investigates current state of the art in multi-sensor data fusion framework. We review research on network traffic measurements and models to understand the self-similar phenomena observed in diverse networking contexts.

In Chapter 3, we outline the problem statement for this dissertation. Two key contributions are addressed, (i) the development of an overlay-based application-aware data transport service framework, and (ii) the design and implementation of multi-sensor data fusion framework.

Chapter 4 explains the design, implementation and performance analysis of the application-aware transport framework. We also illustrate the deployment of an in-network service, i.e., the application aware overlay one-to-many data distribution protocol for multiple heterogeneous CASA end-users.

Chapter 5 presents an architectural framework for multi-sensor data fusion, and reports on the prototype we have built. We also describe our experience with a multi-sensor data fusion application which is implemented using the framework.

Chapter 6 presents a peer-to-peer collaboration framework for multi-sensor data fusion. We propose a data synchronization mechanism and peer selection scheme to coordinate peers for multi-sensor data fusion applications. Simulation-based results show that the proposed scheme can deliver a significant performance improvement, even when the peers have high variability in network and computation resource availability.

In Chapter 7, an analytical mode for end-to-end data fusion latency is proposed. We analyze the data fusion latency considering self-similar network traffic and other application-specific

scenarios. The model is validated by comparing the analytical model with simulation-based results.

Chapter 8, we present the conclusions of this dissertation and outline possible future research.

Chapter 2

BCKGROUND AND RELATED WORK

The recent explosion of interest in distributed sensor networks has led to a wealth of research results in routing protocols, information collection and aggregation, and energy-efficient algorithms in wireless sensor network domain [Ho05]. As the diversity of sensor applications increases, there are new types of sensor applications where high performance sensors are deployed in a large geographical area for environment monitoring, automatic target detection and tracking, battlefield surveillance, and weather monitoring [Cas][Qi01][Ctb]. Because of its flexibility and global accessibility the Internet is used as the preferred medium for communication among the high-performance sensors and various end-users within same or different sensor networks [Ch02][Ch05][Ma04][Cas]. In many sensor applications, critical sensor data has to be distributed to multiple end-users, and the end-users have differing QoS/contents requirements for the data based on the ultimate use of the data. The very basic nature of the sensor applications demands new network services. However, there is a small number of research efforts to provide the application-specific network services for the emerging sensor applications in the current Internet infrastructure.

This chapter provides introduction to CASA, an example of high-performance sensor networks and lists unique data distribution challenges offered by the mission-critical sensor

applications. Since the current Internet architecture may not offer enough functionality and services for such applications, several approaches, e.g., active networks and overlay networks can be considered to provide the required functionality and services for the sensor applications. In this chapter, we point out the existing approaches to address this challenge and the relationship between our application-aware transport service framework and the previous approaches.

Within the Wireless Sensor Network (WSN) domain, simple data aggregation techniques, such as maximum, minimum and average of sensor readings are widely utilized to reduce the overall data traffic to save energy [Ma02]. In the high-end sensing systems such as CASA, it is required to integrate data from multiple sensors using sensor fusion algorithms in order to obtain improved accuracies and more meaningful information [Lim07][Lim08][Qi01][Ja91]. This chapter investigates current state of the art in multi-sensor data fusion. Multi-sensor data fusion processing in a high-end sensor network involves computing intensive signal processing and high-bandwidth communication between the sensors and processing nodes. Furthermore, multi-sensor data fusion applications are usually time-critical. Thus, it is desirable to aggregate computation and communication resources from multiple sensing/processing nodes in the network to meet the execution time requirement of the time-critical applications. In this chapter, we explore resource sharing techniques in a cooperative environment to perform the time-critical data fusion function in a decentralized manner.

Since self-similarity in network traffic is believed to have a significant impact on network performance, networks designed without considering self-similarity of network traffic are likely to function in unexpected ways. This chapter investigates the research on self-similar network traffic including its impact on network performance, and causes of the appearance of self-similarity, and modeling techniques to understand the effect of self-similar cross-traffic on providing QoS for the mission-critical sensor applications.

Section 2.1 provides an overview of CASA and describes the data distribution challenges in the system. In Section 2.2, QoS support schemes in the Internet and various end-host adaptation

schemes are discussed. Section 2.3 describes a variety of active network research and their applications. In Section 2.4, we introduce several proposals in overlay networking. Section 2.5 outlines the existing multi-sensor data fusion frameworks. In Section 2.6, we highlight P2P computing paradigm that enables efficient resource aggregation for high-performance and data-intensive applications to provide acceptable performance. Section 2.7 describes prior work on self-similar network traffic. In Section 2.8, we summarize this chapter.

2.1 CASA - A Distributed Collaborative Adaptive Sensing System

Collaborative Adaptive Sensing of Atmosphere (CASA) focuses on giving more accurate forecasts and early warnings to guard against severe weather patterns, such as hurricanes, tornados and flash floods. Central to the research effort is the use of a large number of small radar systems, spaced closely enough to see close to the ground in spite of the Earth's curvature and to avoid resolution degradation caused by radar beam spreading [Mc05][Zi05][Ku06]. This network of sensors gathers and sends data to processing nodes, which can compute meteorological algorithms that detect, track, and predict weather features. These weather features are fed to end-users, such as government agencies, emergency managers and commercial businesses. The radars and their associated computing and communications infrastructure can be dynamically reconfigured in response to changing weather conditions and end-users' needs. Therefore, the CASA system consists of radar sensors, processing nodes, storage, and underlying network infrastructure for the distributed computation and data transport. Fig. 2.1 illustrates an example of data transport in a CASA network. In the figure, with the control commands and the feedback weather data communicated through the network, the end-users and the sensors form a closed loop. The dotted blue circle illustrates the part of the CASA network that supports communication between the radar nodes and the distributed processing/storage nodes.

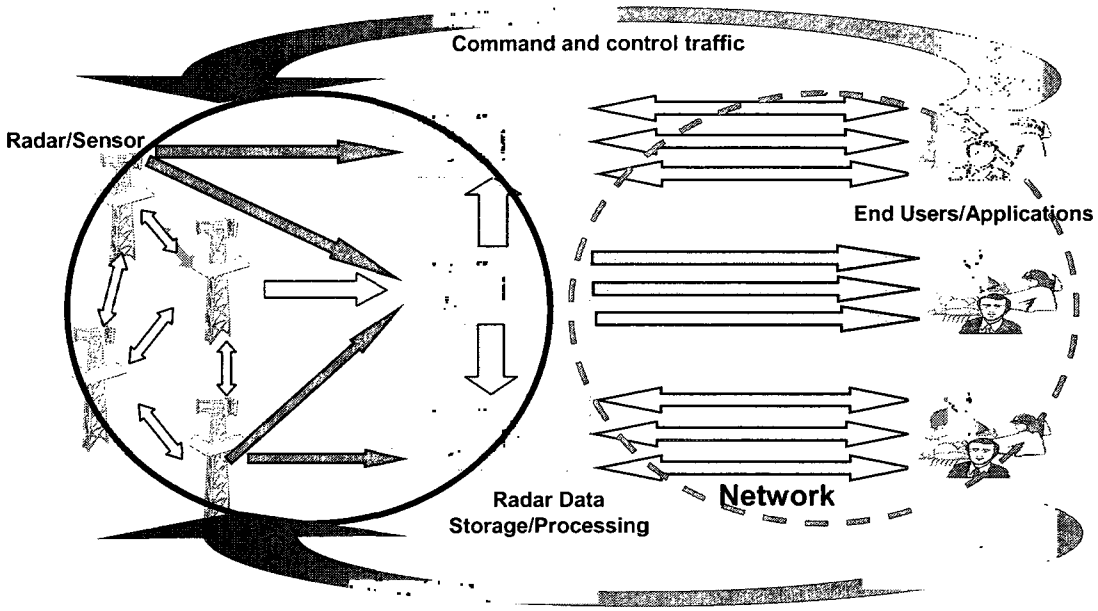


Figure 2.1 Data Transport Requirements in CASA [Ban05]

The dotted red circle highlights the network that provides communication between processing/storage nodes and the end-users/applications. Depending on the resources available at the radar node, data can be locally processed or transmitted over the network (which may consist of both wired and wireless links) in real time for remote processing [Ban05]. In addition, control and command flow must be provided among the distributed nodes.

2.1.1 Data Transport Challenges in a CASA system

In CASA, the data distribution over the network is very challenging due to hostile network environment and the unique data transport requirements of the system. CASA network consists of wired and wireless links, and the network can suffer from wireless link outage or degradation, and dynamic cross traffic, which lead to increasing loss rate and latency. In addition, there are various end-users' requirements depending on the mode of operation. For example, some end-

users' applications are extremely sensitive to network delay, so they may be willing to tolerate data loss to a certain extent instead of recovering the data loss by retransmission. Alternatively, for some end-users, the reliability of data may be more important. One-to-many and many-to-one data transfer/processing scenarios may be required. Unlike audio and video multicast applications in which a source node transmits data to all the receivers with a single send, many end-users of the CASA applications use data generated by same set of sensing nodes in different ways to meeting different system specific goals, e.g., tornado detection and rainfall estimation. In addition, the end-users have different computation capabilities and network conditions. Therefore, the CASA system is required to support a heterogeneous set of end-users with different quality-of-service requirements and access bandwidths at the same time. Moreover, in some cases, the data may only become meaningful when combined. In these cases, many-to-one communication/computation will be required to obtain data from multiple radars for constructing the composite information. Since the CASA applications often deal with massive data set and need a high-capacity and low latency transport service to effectively connect distant locations across a wide-area, the CASA system tends to generate burst traffics, and the burst traffic has adverse effects on other cross-traffic. The same link can be crossed by multiple sensing or command and control flows, periodic bulk data transfers for logging, and third-party wide-area traffic. The interaction among the different traffic in the network can cause congestion and degrade QoS; therefore, it is required to design a transport mechanism to be friendly to the other cross traffic streams on the network while satisfying the requirements of the radar applications. Radar data streaming from the radars to the points of computation also has unique QoS requirements: a minimum acceptable rate based on the end application (CASA needs to support multiple applications with different requirements simultaneously), better accuracy of end results with higher bandwidth, a bound on delivery time beyond which data is not useful, a bound on bursty losses, and more. In addition, the command and control traffic has its own set of QoS requirements. Consequently, we should design, analyze, and implement several new data

transport and computation mechanisms for transmitting radar data between endpoints and processing data to support the time-critical sensor applications in different scenarios: real-time data processing and congestion-controlled data transfer; integrate data from multiple sources, and one-to-many communication approaches to more effectively utilize the aggregate bandwidth from the radars to the processing or storage nodes, and among consumers of radar data.

2.2 End-host Adaptation

Motivated by the end-to-end argument [Sa84] and the principle of Application-Level Framing (ALF) [Cl90], where protocol data units are chosen in concert with the application, end-host application adaptation scheme allows each application at the sender host to decide how best to adapt to network condition independently. As an example of end host adaptation, Odyssey system [No97] exploits adaptation APIs in the context of mobile information access. The system monitors resource levels such as disk space, CPU, and battery power, and notifies applications of relevant changes, and enforces resource allocation decisions. Congestion Manager (CM) [Ba99][An00] system is implemented in-kernel since it has to manage and share resources across applications. The CM maintains congestion parameters and exposes an API to enable applications to learn about network characteristics, pass information to the CM, and schedule data transmissions. In [Gu05], the authors propose an architecture which consists of an input buffer at the server side, coupled with the congestion control mechanism of TCP at the transport layer, for efficiently streaming stored video. The proposed scheme is adaptive to changes in the bandwidth available to the video stream by selecting high priority frames from its source-end. TCP-friendly Rate Adaptation Based on Losses (TRABOL) [Ba02][Ba05], an application-level congestion control protocol is proposed to support Digitized Radar Signal (DRS) transfer in a real-time and high-bandwidth radar data distribution application. TRABOL can decide the network resource

level using its packet loss based congestion control mechanism, and notify a radar application of the current status of network resource, e.g., transmission rate. Based on the resource availability information, the radar application selects a subset of data to achieve better end-user data quality requirements.

However, the network conditions are dynamic and end-hosts cannot predict a change in the available resources at intermediate nodes during the transmission. Therefore the end-host adaptation alone cannot protect the subset of data selected by end-hosts from random losses at the intermediate nodes of network.

2.3 Active Networking for Application-specific Network Services

The basic idea of active networks is to add various forms of extensibility to routers or switches in networks. There are various proposals to support QoS in the Internet using active networking technologies. Active networking offers considerable promise for improving congestion control. Bhattacharjee et al. [Bh97] proposed a programmable congestion control. In the work, the authors consider application-specific congestion control advice for routers that indicates how to reduce the quantity of data at a congestion point in a network. In times of congestion, the routers estimate the amount of data that can be sent over a connection and drop frames of less importance based on the advice. Active Congestion Control (ACC) [Fa98] exploits state and programmability to reduce the delay when congestion is signaled to the sender. ACC packets contain characterizations of the state of the endpoint's congestion feedback. When congestion occurs at a router, the router determines the congestion window size, deletes packets that would not be sent with this new window size, and informs the sender of the new window size. Thus, ACC is particularly powerful in networks that have large bandwidth-delay products. In [Me99] the authors presented AMnet which addresses group communication and aims at providing scalable

heterogeneous multicast where participants can individually select service level. More recently, Keller et al. [Ke00] proposed an active router architecture for multicast video distribution. They use a wavelet-based adaptive video codec called WaveVideo[Fa99][Ke00], which allows high level scaling the stream. In the WaveVideo, a two-dimensional wavelet transform is applied to an image, and split the image into a low-frequency and three high-frequency sub-bands. This scheme enables router in the network to drop packets from high-frequency sub-bands first when there exists a high load situation of the outgoing link, since the loss of high-frequency sub-bands results in only a slight degradation of the quality. Each of the packets in the system carries a WaveVideo-specific tag, which allows for reduction of the stream's bandwidth by selectively dropping packets belonging to a particular quality set. Calvert et al. [Cal01] propose CONCAST service that provides a general form of scalable many-to-one channel. CONCAST offers a solution to a problem arising in some group communication, where multiple sources send messages toward one destination. In contrast to multicast services, which enable one sender to reach a large number of receivers via a single sending operation, CONCAST delivers a single merged copy to a destination. Most of the work in active networks requires the infrastructure support in order to provide extensibility. Due to the requirements of change to the network infrastructure, only few network services based on active networking are deployed in real network.

2.4 Overlay Networks for QoS Support and Application-specific Services

Overlay networks have been proposed to provide a range of useful services for enhancing Internet applications including better end-to-end routing [An01][Su02], multicast [Ch00][Ca03][Ko03] without requiring modifications to the basic Internet infrastructure. Several routing overlays have been proposed to enhance performance, robustness, and functionality of the Internet routing.

Detour [Sa99] points out several routing inefficiencies of the current Internet. Moreover, Detour envisions an overlay network based on IP tunnels to prototype new routing algorithms on the top of the existing Internet. Resilient Overlay Network (RON) [An01] is an architecture that is proposed to quickly detect and recover from path outages and degraded performance. In addition, RON allows applications to prioritize some metrics over others (e.g., low-latency, high-throughput, etc.), so the integration routing and path selection with the applications is tighter than the traditional IP routing's. However, RON is limited to a small number of participating nodes and cannot scale to the number of ASes (Autonomous Systems) that exist today, because of its fully connected mesh topology and probing overhead. To address the scalability problem of RON-like routing overlay networks, Nakao et al. [Na06] propose a system, called PLUTO. Instead of using a fully connected mesh, PLUTO uses a low-cost approach to building a topology-aware routing mesh that eliminates virtual links that contain duplicate physical segments in the underlying network. In [Liu04], the authors propose application level relay schemes for high-bandwidth data transport. By optimally combining TCP connections, both sequentially and concurrently, their proposal greatly improves the throughput of long-haul data transport over lossy links.

Overlay multicast is one of the promising applications for distributing content to multiple end-users over a wide-area. The goal of Overcast [Ja00] is to provide wide-area content distribution and bandwidth sensitive multicast services while utilizing the network bandwidth efficiently. Bullet [Ko03] is another high-bandwidth data dissemination overlay. Bullet consists of a tree with a mesh overlaid on top of it. The data stream is divided into blocks which are further divided into packets. Nodes transmit a disjoint subset of the packets to each of their children. Nodes receive a subset of the data from their parents and recover the remaining by locating a set of disjoint peers. SplitStream [Ca03] is another tree-based streaming system that is built on top of the Scribe [Ca02] overlay network. In SplitStream, the data is divided into several disjoint sections called stripes, and one tree is built per stripe. In order to receive the complete

stream, a node must join every tree. A node is only responsible for data forwarding on one of the stripes, so if a node suddenly leaves the system, at most one stripe is affected. Those overlay multicast protocols are explicitly designed with the goal of building distribution trees that maximize each node's throughput from the source. However, they do not consider the issues associated with real-time transmission of the sensor network applications.

For real-time applications, such as conferencing, several overlay based multicasting schemes have been proposed. Narada [Ch00] constructs an overlay structure among participating end systems in a self-organizing and fully distributed manner, making use of network metrics like bandwidth and latency. By leveraging a bandwidth-delay based routing algorithm, they can construct overlay trees simultaneously optimized for bandwidth and latency. In [Ba03], the authors present Overlay Multicast Network Infrastructure (OMNI) to efficiently implement large-scale media-streaming applications on the Internet. OMNI consists of a set of Multicast Service Nodes (MSN) and a set of end hosts. MSN provides efficient data distribution services to the end hosts, and an end host subscribes with a single MSN to receive multicast service. These MSNs are organized into an overlay using a decentralized manner. As demonstrated in the examples above, the basic issue in overlay-based multicast is how to establish and maintain the topology of multicast distribution trees. However, for the emerging sensor applications, the prevailing receiver heterogeneity in capabilities and preferences should be explored. We believe that integrating application-aware adaptations with the overlay-based topology management techniques above will greatly facilitate to produce a fully adaptive multicast solution.

Most of the overlay network literatures have focused on proposing overlay routing functionalities or application-specific solutions such as high-bandwidth multicast for content distribution, but Li and Mohapatra [Li04] proposed a general unified overlay framework for quality of service guarantees. They introduced the concept of overlay brokers (OB), and designed QoS aware Routing protocols for Overlay Networks (QRON). The goal of QRON is to find a QoS-satisfied overlay path, while trying to balance the overlay traffic among OBs and overlay

links. Two other efforts that share similar goals of building a generic overlay network are OverQoS [Su02] and SON [Du02]. Using OverQoS, overlay network service providers can offer QoS to the customers using Controlled Loss Virtual Link (CLVL) technique, which ensures that the loss rate observed by aggregation is minimal as long as the aggregate rate does not exceed a certain value. OverQoS can be employed to provide Internet QoS such as differentiated rate allocations, statistical bandwidth and loss assurance, and can enable explicit-rate congestion control algorithms. In addition, OverQoS can provide a variety of QoS enhancing services to applications including: (i) eliminating the loss bursts by smoothing packet losses across time; (ii) prioritizing packets within an aggregate; (iii) statistical bandwidth and loss guarantees. Service Overlay Networks (SON) is designed to use overlay technique to provide value-added Internet services. A SON can purchase bandwidth with certain QoS guarantees from ISPs to build a logical end-to-end service delivery overlay. The authors have formulated the problem of QoS provisioning considering various factors like SLA (Service Level Agreement), QoS requirements, traffic demand distribution and bandwidth cost. Yoid [Fr00] is a generic overlay architecture which is designed to support a variety of overlay applications that are as diverse as net-news, streaming broadcasts, and bulk email distribution.

OCALA [Jo06] and Oasis [Ma06] enable the users of legacy applications to leverage overlay functionality without any modifications to their applications and operating systems by providing an additional layer and APIs. The above proposals focus on using overlay networks for general purpose packet delivering, on the other hand our work addresses the deployment of a variety of application-specific network services in overlay nodes in order to support application-aware data transport. X-Bone [XBONE] is a system for automated deployment of overlay networks. It operates at the IP layer and is based on IP tunnel technique. The main focus is to manage and allocate overlay links and router resources to different overlays and avoid resource contention among the overlays. A similar approach was proposed in OPUS [Br02], which provides a large scale common overlay platform and the necessary abstractions to service multiple distributed

applications. It automatically configures overlay nodes to dynamically meet the performance and reliability requirements of competing applications. While our framework focuses more on programming of a single overlay node, OPUS focuses on the wide-area issues associated with simultaneously deploying and allocating resources for competing applications in a large scale overlay networks.

In this dissertation, we explore an overlay based approach in order to provide the required networking services for the mission-critical sensor applications as overlay based approaches can provide immediate solutions to deploying new network services without requiring the modification of the current Internet architecture. Furthermore, our application-aware data transport service framework can be combined with recent overlay routing techniques mentioned above to provide robust communication services for the end-users.

2.5 Multi-sensor Data Fusion

Data fusion has emerged as a crucial building block in designing a sensor network. In wireless sensor networks, the primary goal of data fusion is to eliminate redundant transmissions. Madden et al. [Ma02] discuss the implementation of five basic database aggregates, i.e. COUNT, MIN, MAX, SUM, and AVERAGE, based on the TinyOS platform and demonstrate that such a generic approach for aggregation leads to significant energy savings. Other previous works [In00], [Shr04] in the related area aim at reducing the energy expended by the sensors during the process of data gathering. Directed diffusion [In00] is based on a network of nodes that can co–ordinate to perform distributed sensing of an environmental phenomenon. Such an approach achieves significant energy savings when intermediate nodes aggregate responses to queries. The SPIN protocol [He99] uses meta–data negotiations between sensors to eliminate redundant data transmissions through the network. While many of the proposals have used static role assignment

approaches, DFuse [Ku03] supports distributed data fusion with automatic management of fusion point (where aggregation occurs) placement and migration to optimize a given cost function (such as network longevity). A mobile agent based multi-sensor data fusion infrastructure is proposed in [Qi01a]. Compared with the traditional client/server paradigm, mobile agent adopts a new computing model: data stay at the local site, while the execution code is moved to the data sites. Mobile-agent-based multi-sensor fusion saves the network bandwidth and provides an effective way to overcome network latency. It is used to integrate preprocessed data located at local sensor nodes. As larger amount of sensors are deployed in harsher environment, it is important to integrate data from multiple sensors to handle uncertainty and faulty sensor readings. Qi et al introduce several fault-tolerant fusion algorithms which are used to provide error tolerance using the redundancy in the sensor readings [Qi01b].

Unlike resource limited wireless sensor networks, which are designed to carry out one or few tasks while minimizing energy expenditure, DCAS systems are intended to serve a variety of applications and end-users [Li07]. The data generation rate at each of the sensors in a DCAS system can be several Mbps to tens of Mbps. Because the massive amount of data generated by such sensors and the computing intensive processing required by applications, these sensor/processing nodes are usually equipped with significant communication and computation resources. As illustrated in Section 2.1, DCAS systems consist of high bandwidth sensor nodes and powerful processing nodes in a resource-rich environment. A key design consideration of the DCAS network is the ability of the sensor network to meet the application-specific real-time requirements, while optimizing resource usage. In the high-end sensor applications, such as CASA, geographically distributed sensors generate substantial volume of data, thus it is required to use intensive communications for gathering correlated data from multiple sensors. In addition, the large volume of generated data must be processed with stringent real-time constraint.

2.6 P2P Networks for Resource Aggregation

In such resource-demanding circumstances of multi-sensor data fusion, a distributed dynamic collaboration approach built on peer-to-peer (P2P) architecture is of interest to aggregate unused resources from multiple sensor/processing nodes across the network. In file sharing P2P networks, such as BitTorrent (http://www.bittorrent.com), Napster (http://www.napster.com) and Gnutella (http://www.gnutella.wego.com), each peer supplies a disk space to store the community's collection of data and bandwidth to move the data to other peers, and obtains others' resources in return. When communication costs become too high (e.g., in case of large multimedia content transfers), P2P networks can avoid the bandwidth limitation problem by spreading the cost over several peers. Apart from the traditional file sharing, a significant amount of research has been done on the development of P2P architecture to provide access to remote resources for high-performance and data-intensive applications, such as large-scale scientific and engineering research applications. Such P2P computing aggregates the unused computer cycles that are scattered on the network to support distributed computing [Ya06][INTEL][Lo04]. Some of the projects focus on harnessing the idle computing resources of personal computers to solve scientific problems [Ko01][An04][SET][CON].

Motivated by the needs and success of those projects, such as SETI@home[SET] and genome@home[GEN], Awan el al. [Aw06] propose an architecture for distributed cycle sharing in large-scale peer-to-peer environment. The dynamic nature of P2P networks makes it difficult to build and maintain structured networks and to use state-based resource allocation techniques. They build a P2P based system to work in an unstructured P2P environment similar to current file-sharing networks such as Gnutella and Freenet. In doing so, they are able to leverage vast network resources while providing resilience to random failures. The use of the unstructured P2P-like mechanism in BPS is motivated by the success of real-world file sharing networks, such as Gnutella. A recent proposal in [Ya06] strives to build an open infrastructure that support higher

quality of service and fault tolerance for processing dead-line driven tasks in a P2P environment. In their scheme the selection of peers is based on a joint evaluation of peer computational availability and communication bandwidth to process a job prior its deadline, and this peer selection scheme is similar to BPS. A simple resource aggregation approach in P2P network may not be enough for multi-sensor data fusion in the sensor application. In the multi-sensor data fusion, data needs to be combined in such a manner that the real-time constraints of the sensor application, as well as application-specific data selection requirements are met. There is need for a framework to support diverse data fusion considering the unique requirements of mission-critical sensor applications.

2.7 Network Self-similarity and its Effect on Network Performance

The main objective of this dissertation is to provide the desired QoS for mission-critical DCAS end-user applications. Understanding the offered network traffic properties on the network is the first step to provision the infrastructure that complies with the QoS requirements of the end-users. This section introduces network self-similarity which is an important notion in the understanding of network traffic including modeling and analysis of network performance. In the past several years, measurements of local area [Le94] and wide area traffic [Pa95] have shown that network traffic possesses self-similar property; the measured time series is bursty across several time scales. In [Le94], Leland et al. reported the results of a massive study of Ethernet traffic and demonstrated that it exhibits self-similar property or long-range dependence (LRD). There are several works that attempt to explicate the physical cause of self-similarity in network traffic. Park et al. [Pa96] have shown that the transfer of files whose sizes are drawn from a heavy-tailed distribution is sufficient to generate self-similarity in network traffic. Specifically, by measuring self-similarity via the Hurst parameter H and the file size distribution by its power law exponent α,

it has been shown that there is a linear relationship between H and α over a wide range of network conditions. The on/off model developed by Willinger et al. [Wi97] establishes that the aggregation of a large number of on/off sources with heavy-tailed on and off periods cause self-similarity at multiplexing points in networks. Research done in [Cr97] has revealed that the traffic generated by World Wide Web transfers shows self-similar characteristics. The distribution of file sizes in the Web might be the primary determinant of Web traffic self-similarity. In [Pa97], the same authors demonstrated performance implications of self-similarity in terms of packet loss rate, retransmission rate, and queuing delay. Based on a simulation study, they show that increased self-similarity results in performance degradation. In [Ji01], the authors investigate the impact of self-similarity on wireless data networks. Using simulations with genuine traffic trace which exhibit long-range dependent behavior, they show that the self-similar traffic traces, compared to traditional traffic models such as the Poisson model, produce longer queues.

Mathematically, a continuous-time random process *X(t)* is said to be self-similar with Hurst parameter $H \in [0,1)$ if for all $a > 0$ and $t \geq 0$, [Pa00]

$$X(t) = a^{-H} X(at) \qquad (2.1)$$

The Hurst parameter H is a measure of self-similarity of the continues-time random process. A self-similar time-series possesses a hyperbolically decaying auto-correlation function of the form,

$$r(k) \approx k^{(2H-2)} L(t), \text{ as } k \to \infty, \qquad (2.2)$$

where *L(t)* is a slowly varying function at infinity [Le94]. Therefore, the auto-correlation function is

$$\sum_k r(k) = \infty. \qquad (2.3)$$

This infinite sum is the definition for long range dependence, so all self-similar processes are long-range dependent. The traditional traffic models based on Poisson process or, more generally, short-range dependent (SRD) processes, cannot describe the behavior of network traffic [No94].

Norros [No94][No95] suggests a self-similar network traffic model based on fractional Brownian motion (FBM) as a feasible and attractive alternative to the traditional traffic model. The FBM process describes significant features of the real traffic using three parameters *m, a* and *H* with the following interpretation [No94]. *m* is the mean rate (or equivalently, resource utilization) that measure the volume or quantity of traffic. The other two parameters refer to the burstiness or quality of traffic. *a* (the peakedness) measures the magnitude of fluctuation around the mean rate *m*. At the unit time scale, it is the ratio of the variance of traffic volume to the mean value. *H* (the Hurst parameter) is an indication of rate of decay of correlations in the traffic. In traffic arrival model, an FBM input process is represented by

$$A(t) = mt + \sqrt{ma}Z(t), t \in (-\infty, \infty) \qquad (2.4)$$

where *Z(t)* is a normalized FBM process with the following properties: (a) *Z(t)* has stationary increments; (b) *Z(0) = 0*, and *E[Z(t)] = 0* for all *t*; (c) $Var\{Z(t)\} = t^{2H}, H \in [\frac{1}{2}, 1)$, and (d) *Z(t)* has continuous a sample path. In a queue with an FBM input process, the asymptotic queue length distribution can be estimated by approximation [No94][No95]. Consider an infinite buffer queue with a constant service rate C. Let $A = (s,t]$ be the amount of traffic that arrives at the queue over the time interval $(s,t]$, and let $A(t) = A(-t,0]$. The queue length at time 0 is

$$Q = \sup_{t \geq 0}(A(t) - Ct). \qquad (2.5)$$

and the probability that the queue length exceeds x is

$$P\{Q > x\} = P\{\sup_{t \geq 0}(A(t) - Ct) > x\} \qquad (2.6)$$

The above expression is very difficult to evaluate, thus, most of the previous studies use the lower bound [No94][No95][Ji06][Fr03]

$$\begin{aligned} P\{Q > x\} &\geq \max_{t \geq 0} P\{(A(t) - Ct) > x\} \\ &= \max_{t \geq 0} \overline{\Phi}\left(\frac{(C-m)t + x}{\sqrt{amt^H}}\right) \end{aligned} \qquad (2.7)$$

where $\overline{\Phi}$ is the complementary distribution function of the standard normal distribution.

By differentiating, the maximum in (2.7) can be obtained at

$$t = t^* = \frac{H}{1-H} \cdot \frac{x}{C-m} \tag{2.8}$$

The time-scale t^* is the Maximum Time-Scale, and it is a point in time where the unfinished work in the queuing system achieves its maximum in a probabilistic sense [Fo00].

Using further approximation [No94]

$$\overline{\Phi}(y) \approx \exp(\frac{-y2}{2}),$$

The following expression can be obtained [No94][No95]

$$P\{Q > x\} \geq \overline{\Phi}\left(\frac{(C-m)^H x^{1-H}}{\kappa(H)\sqrt{am}} \right) \sim \exp\left(-\frac{(C-m)^{2H}}{2\kappa(H)am} x^{2(1-H)} \right). \tag{2.9}$$

where $\kappa(H) = H^H (1-H)^{1-H}$.

The basic approximation in (2.7) is the Chernoff bound for the RHS of (2.9).

$$\overline{\Phi}(y) \leq \exp(\frac{-y2}{2}).$$

It is known that the Chernoff bound is tight for sufficiently large y. However, for small y, this may not be the case [Gu06]. In fact, simulations show that the basic approximation generally provides a reasonable upper bound for (2.7), but no proof of this has been presented [Man02]. The basic approximation is useful in making rough performance estimates for provisioning network to support a desired level of quality of service (QoS), e.g., loss rate and network delay. However, most traffic in real networks has a more complicated scaling behavior, which cannot be described by a single set of parameters. In order to capture such traffic features more precisely, multi-fractal stochastic models are proposed. [Ji06] presents a multi-scale traffic model that incorporates the notion of a piecewise self-similar process to analyze long-range dependent traffic

with changing scaling exponent, i.e. Hurst parameter over time scales. A similar approach is proposed in [Fr03]. The authors develop a model known as two-scale Fractional Brownian Motion which captures the observed Gaussian behavior of Internet backbone traffic. Using the model, the authors also develop a procedure to compute the end-to-end delay across a network.

The previous analysis using FBM model can be extended for sensor network performance analysis. In this dissertation, we use the FBM based queuing analysis to estimate the end-to-end fusion latency in Internet-based sensor networks.

2.8 Summary

This chapter provided an overview of CASA and described the data distribution challenges in the system. We then presented QoS support mechanisms in the Internet, such as end-host adaptation, active networks and overlay networks. We make an argument in favor of exploiting overlay-based approaches for providing the required functionalities and network services for the emerging high-end sensor applications.

Data fusion is becoming common in wired/wireless sensor network domains. This chapter reviewed the existing work in the sensor fusion from three aspects: reason for sensor fusion and existing multi-sensor data fusion frameworks, data processing paradigm focused on P2P computing, and self-similarity in network traffic which can impact the performance of data fusion applications. Based on the previous work, we develop a multi-sensor data fusion framework that can meet the QoS requirements of the sensor applications.

Chapter 3

PROBLEM STATEMENT

A new class of DCAS (Distributed Collaborative Adaptive Sensing) systems is emerging that relies on collaborative operation among networked sensors, computation nodes and end-users [Cas][Mc05][Zi05]. In many of these systems, sensors gathering data or information, multiple distributed processing nodes and end-users adaptively collaborate in a tightly coupled loop. The features of the new class of sensor networks can be summarized as follows: a) sensors, processing nodes, storage units and end-users are distributed throughout a physical space, and are connected with wired or wireless networks; b) the availability of underlying network resources can fluctuate dynamically; c) there are various communication needs for the DCAS data distribution (e.g., real-time, reliable or unreliable, and high or low bandwidth); d) in the collaborative sensing environment of DCAS systems, data from multiple sensors are combined in order to achieve enhanced accuracies and fault-tolerance; and e) the fusion processing has to be done in real-time.

As mentioned in Chapter 1, the Internet is being used as a medium for communication in this class of sensor networks. The basic best-effort architecture of the Internet poses challenges to cope with the challenges in implementing the sensor networks and this type of communication is unlikely to change. In this dissertation, we present an overlay-based application-aware data transport service framework that can enable new service implementation in the network without changing infrastructure.

Data fusion in the mission-critical high-speed sensor applications such as CASA must employ resource-efficient data processing mechanisms and robust communication to acquire acceptable quality and real-time processing requirements. This dissertation develops a data fusion framework to support time-critical and resource demanding multi-sensor data fusion applications. Self-similarity in network traffic observed in diverse networking contexts is believed to have a significant impact on network performance [Pa96][Pa97]. Our research includes investigating the impact of self-similarity in network traffic on the performance of multi-sensor fusion applications.

In the rest of this chapter, we state the research goals, objectives in pursuit of this novel approach to design and develop the frameworks.

3.1. Research Goals

Our ultimate goal of this research is to design and develop software frameworks that support data distribution/processing in high-end sensor networks, and demonstrate the effectiveness of the proposed frameworks. Toward the end, we design and implement (i) an application-aware transport service framework, and (ii) a multi-sensor data fusion framework. The key requirements of the frameworks are to:

- concurrently meet the various data streaming requirements of the DCAS end-users.
- allow applications to adapt to dynamic network conditions using application-aware in-network services to best meet the QoS requirements of the applications.
- facilitate the development and the deployment of the application-aware in-network services.
- provide an abstraction layer to ease the implementation of various multi-radar data fusion applications.

- meet the real-time requirements of multi-sensor data fusion processing.
- offer a time-efficient data managing and searching mechanism for data fusion applications in CASA.

3.2. Research Objectives

3.2.1 Application-aware Transport Service Framework

To support the mission-critical DCAS applications the framework needs to enable the applications to respond to the resource availability. This dissertation implements a software framework for DCAS data distribution over the Internet. The specific assumptions for the framework are as follows: a) many end-users can operate with varying QoS within bounds for data quality and bandwidth. Thus, scaling of the sensor data can be performed to optimally adapt to different transmission rate over a wide range of QoS levels; b) in-network application-aware processing can be used to control the adaptation to best meet content, task, and end-user specific requirements; c) the application-aware processing can be deployed in the intermediate nodes of the overlay network, and d) the overlay networks can provide the required connectivity and transport quality to the applications according to the application requirements. Based on the assumptions, this section identifies key objectives that have to be achieved by the application-aware transport service framework to realize the goal of this dissertation.

3.2.1.1 Objectives for Application-aware Transport Service Framework

The main objectives of the application-aware transport service framework are listed below:

1) Design and implement an architecture and an API for the DCAS data distribution. The

framework has to provide an application-friendly mechanism for application programmers, so the application programmers utilize it to implement the service plug-ins, as well as to deploy the developed plug-ins on the overlay nodes. In designing the architecture and the API, the following factors are considered:

- Usability: To be widely accepted, the amount of effort required by the application programmers to make use of the service should be minimized. Therefore, it should not be necessary for the application developer to have any prior knowledge about the internal details of overlay networks and the framework. In addition, the framework should provide a clean and straightforward interface between the application-aware services and the architecture.
- Extensibility: The API should also be generic enough to support diverse application-aware services and the requirements of applications, so that a variety of different application-aware services can be implemented using the same API and architecture.
- Deployability: The architecture should enable new application-aware services to be deployed in the overlay nodes with simple reconfiguration. In addition, the deployment of the new application-aware services should not worsen the performance of other existing application-aware services.

2) Demonstrate the effectiveness of the framework by implementing new application-aware services using the API. We implement two key services to show the ease of the framework:

- Application-aware selective data drop/forwarding: High-bandwidth sensor applications can potentially benefit from a combination of end host and in-network processing. Application-aware selective drop and forwarding can enhance the QoS provided to the applications by allowing the applications select

packets based on their own criteria at end hosts or intermediate nodes when they encounter resource fluctuation.

- Application-aware heterogeneous multicasting: End-users joining the multicast session can have different quality requests due to the variability of the network bandwidth or the processing capabilities of the end-hosts. Multicasting to heterogeneous end-users is a service that can benefit from in-network processing offered by our framework. At intermediate multicast nodes, new streams with different qualities can be derived from the received ones in an application-aware manner, and hence the diverse quality requirements of heterogeneous clients can be satisfied.

3.2.2 Multi-sensor Data Fusion Framework

Multi-sensor data fusion applications are becoming common in the large scale collaborative sensing systems [Li07][Lim07]. With multi-sensor data fusion applications, end-users/applications need to pull out data to be integrated from multiple sensors and processing nodes. Since substantial amount of data has to be collected from geographically distributed locations, and processed in real-time, efficient use of available computation/communication resources in the sensor network is very important. Another critical issue to be considered is to select a set of data from multiple sensors when the sampling time and data generation intervals of the sensors are not synchronized. One of the key goals of this dissertation is to develop a multi-sensor data fusion framework. In this section we identify the key objectives of the multi-sensor data fusion framework.

3.2.2.1 Research Objectives for Multi-sensor Data Fusion Framework

The main objectives of this research are listed below:

1) Design and develop a multi-processing architecture for real-time multi-sensor data fusion. The multi-sensor data fusion architecture provides an abstraction layer for various multi-radar data fusion algorithms to offer ease of programming.
2) Development of a multi-process coordinating mechanism distributes computation and communication tasks required by the algorithms to meet execution time requirements. The framework needs to accommodate simultaneous and collaborative operation of multiple processes over a single node or multiple nodes across the sensor network.
3) Development of a time-efficient data selection mechanism. Due to the need to merge data, it is necessary to select a set of data items collected from an area scanned by the multiple radars. Various multi-sensor fusion applications have different data selection criterion in terms of the minimum number of data from different sensors for acceptable final result, the maximum possible waiting time for data when the number of data is not sufficient, and finally the maximum tolerance in sampling time difference among the data to be integrated. Thus, the common volume module needs to select a set of data considering the application-specific requirements.
4) Demonstrate the effectiveness of the framework by implementing CASA multi-radar data fusion algorithms using the framework. Evaluate the effectiveness of the framework through various experiments using data collected in CASA IP1 network.
5) Development of an analytical model for data fusion latency. One of the objectives in developing the multi-sensor data fusion framework is to propose a model to predict the time for acquiring data to be integrated from multiple distributed sensors considering self-similar network cross-traffic that can affect the performance of the sensor network.

Chapter 4

APPLICATION-AWARE DATA DISSEMINATION USING OVERALY NETWORKS: AN ARCHITECTURE AND PROGRAMMING INTERFACE

The Internet has become a transport medium for real-time critical applications such as remote surgery, sensing and video conferencing. Many of these applications are sensitive to the quality of the data delivered to the end-users. Moreover, each application may have multiple data quality requirements that need to be met for their reliable operation. For example, in Distributed Collaborative Adaptive Sensing (DCAS) systems, where volumes of mission-critical data have to be distributed to multiple users at distant geographical locations in real time, rely on the Internet, a best-effort network, for connectivity [Mc05]. Within DCAS, data streams and users may have differing QoS requirements for the data based on the nature and the ultimate use of the data. In addition, different users may need different subsets of data from the total data set generated by sensing nodes in a given period of time.

In a best-effort network, meeting heterogeneous QoS requirements of multiple end-users become even more challenging under dynamic resource fluctuations within the network. Here resources refer to capacity, available bandwidth, buffer space etc., that affect the performance of a stream of data passing through, as well as connectivity, redundant paths for fail-safe operation, etc., that affect the reliability of the application. This motivates the need for an end-to-end adaptive networking infrastructure. The key philosophy about adaptive network infrastructure is

that both network and applications adapt their operations to best serve the QoS requirements of the end-users. In the context of DCAS systems, as described below, QoS requirement extends beyond delay and bandwidth provisioning, to include properties such as fail-safe operation. This paradigm of adaptive networking is critical to the deployment of mission-critical DCAS systems over the best-effort Internet. These mission-critical DCAS systems need to keep operating even when a phenomenon such as a tornado brings down the part of the network infrastructure. However, the current Internet is often not able to offer desired QoS support and resilience for the DCAS applications. Several approaches, e.g., in-network processing using active networks [Te97] and overlay networks based approaches have been proposed to deal with the current limits of the Internet architecture. Overlay networks provide ways to mitigate the current limitations of the Internet by enabling applications to make application-aware adaptation to meet the QoS requirements of the end-users. For example, overlay networks may allow applications to make routing decision at the application layer about the next hop to be taken to meet the latency and connectivity requirements during link failure, and even allow it to use multiple service providers to meet overall operational requirements. Similarly, overlay networks allow applications to perform application-aware congestion control so that performance of the application does not degrade significantly during network congestion. Alternatively, active networks introduce the concept of in-network processing where intermediate nodes such as routers and switches of the network perform customized computations on messages being forwarded in order to effectively support application-aware adaptations [Ca06][Te97]. However, to be deployed in the current Internet, the active networks need fundamental changes to the Internet infrastructure, especially at lower network layers. As an alternative, the concept of active service is introduced which does not require changes to the key Internet infrastructure by restricting the location of application-specific computation to the application layer [Am98]. This enables active services to be deployed incrementally in today's Internet. However, most of these active networking and active service frameworks have focused on single network node processing and explicitly avoided saying much

about the end-to-end architecture [Ca06]. Unlike active services, the overlay networks have been proposed as a way for enhancing end-to-end service availability, performance, and scalability [An01][Ba03][Zh04]. Overlay networks can select a route(s) that is intrinsically tuned to the specific needs of applications, e.g., latency over throughput, or low loss rate over latency [An01][Li04][Zh04]; however, most of the current overlay network based solutions do not permit individual applications to determine how best to adapt to the resource availability and user requirements [Br02]. Applications relying on overlay-based implementations to achieve performance, reliability and other application specific requirements must be able to configure overlay nodes to perform in-network application-aware processing. A flexible, efficient approach for the deployment of QoS-sensitive applications using overlay networks should facilitate the monitoring of the QoS received by an application in the overlay network, and allow easy deployment of application-aware processing at intermediate overlay nodes. A framework is thus required for realizing such application-aware overlay networks.

This chapter presents an Application-aWare Overlay Network (AWON) framework for the development of overlay network based adaptive distributed network application. API is proposed to facilitate the implementation and deployment of the application-specific in-network processing capabilities on overlay nodes within the AWON framework. One of the key highlights of the AWON architecture is that in this framework each participating overlay node can be configured to perform different application-aware operations to meet the overall goals of the application(s). In order to realize AWON based implementations, API must support node configuration in an application-aware manner, with each node being configurable to support multiple applications concurrently. There is also a need for communication between the application and the underlying overlay layers for supporting application-specific QoS requirements [An02][Ba06][Lee06]. For this to be realized, the API must allow an application to specify its QoS requirements to the system. When the underlying system is able to accept the application with its QoS requirements, the API should be able to communicate this acceptance to the application [Ba07].

We demonstrate the effectiveness of the AWON framework by considering an emerging distributed collaborative adaptive system, Collaborative Adaptive Sensing of the Atmosphere (CASA) [Mc05]. CASA is based on a dense network of weather radars that operate collaboratively to detect tornadoes and other hazardous atmospheric conditions. In a CASA application, high-bandwidth weather radar data is distributed to multiple users with heterogeneous QoS requirements. The API is used to implement and to deploy the application-aware services plug-ins in the source nodes, the intermediate nodes and the multicast nodes of an overlay network. The real benefit of AWON is to facilitate the deployment of various application-aware services on overlay networks. We also measure the overhead of AWON to show that the overhead is acceptable. The AWON architecture and the experimental results presented in this chapter are based on [Lee06] and [Ba07].

A significant amount of research has been done on the design and development of overlay routing protocols to improve an underlay network's resilience and performance [An02][Ba03][Zh04]. Our work complements and takes the advantages of such ongoing research effort of performing QoS-aware routing in overlay networks. OverQoS[Su04], an overlay-based architecture can provide a variety of QoS-enhancing in-network services in the intermediate nodes of overlay networks, such as eliminating burst losses, prioritizing packets within a flow, and statistical bandwidth and loss guarantees. Our work is motivated by the same vision of enhancing QoS support within the network without the support from IP routers. An important difference between the AWON and the OverQoS architectures is that in the AWON-based approach, quality of service provided to an application is enhanced by performing application-aware processing within the overlay network. Moreover, the AWON architecture is highly flexible and can accommodate QoS requirements of a broad class of applications. OCALA [Jo06] and Oasis [Ma06] enable the users of legacy applications to leverage overlay functionality without any modifications to their applications and operating systems.

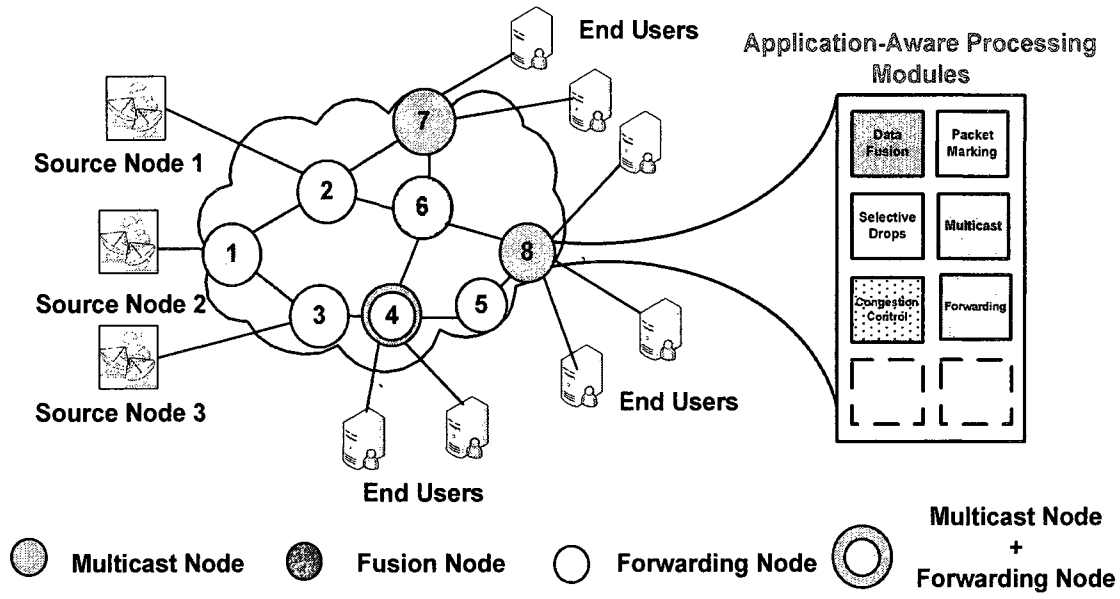


Figure 4.1 Overlay network for application-aware data dissemination

Opus [Br02], which is motivated by active networking, provides a large-scale common overlay platform and the necessary abstractions to service multiple distributed applications. In contrast to our work, Opus focuses on the wide-area issues associated with simultaneously deploying and allocating resources for competing applications in a large-scale overlay networks.

The rest of this chapter is organized as follows. Section 4.1 explains the AWON architecture for deploying application-aware services in overlay networks. Section 4.2 describes the API. An example implementation is illustrated in Section 4.3. Section 4.4 presents experimental results that demonstrate the effectiveness of the AWON and the corresponding API for weather radar data streaming. The summary of this chapter is presented in Section 4.5.

4.1 Application-aware Overlay Networking Architecture

Fig. 4.1 shows an application-aware overlay network for distributing data to multiple sink nodes with different end-user requirements such as data quality and bandwidth requirements. Let us now illustrate the myriad roles overlay nodes may play in meeting application requirements. In Fig. 4.1 source nodes 1-3 may perform application-level packet-marking to indicate the usefulness of the data to a particular application; forwarding nodes (nodes 1-5) may perform packet forwarding/drop based on the marking done by the source nodes; multicast nodes (nodes 4, 7, and 8) may distribute data to multiple users and perform independent congestion control for each user in an application-aware manner. Note that the multicast nodes combine the requests from the users and send an aggregate upstream request to the specific source node, and on receiving data from source nodes forwards appropriate segments of data to the different users; a multicast node in this context thus performs request aggregation and data selection functions beyond that normally associated with traditional form of multicasting. If the network experiences congestion, congestion-based packet or information discard may be performed at the source or at intermediate nodes, according to the available bandwidth. A source node can thus mark packets based on the relative importance of the information sent to the multicast nodes 4, 7, and 8. This facilitates application-aware selective drops rather than random drops within the network. Intermediate forwarding nodes 1-5 may use this marking information at the time of forwarding during network congestion. Similarly node 6, a fusion node, may combine data from multiple sources to reduce the downstream data bandwidth requirements. In addition to the various packet handling functions discussed above, there are two other classes of functions a node may implement. First, there is a need to interface with overlay networks to send and receive packets. Second, it is desirable to run multiple applications simultaneously on the same overlay network. In order to provide those functions, we design four key components of the AWON: *(i) Application Programming Interface, (ii) Application Service Plug-ins (ASP), (iii) AWON Proxy, and (iv) Application Manager.* The application programming interface (API) facilitates development of application-aware services within the architectural framework, and enables

communication between the applications and the overlay routing protocol. Using the API, an Application-specific Service Plug-in (ASP) implements the functions performed by the overlay nodes participating in data forwarding. The main roles of the AWON Proxy (AP) are to provide ASPs with functions for sending/receiving packets to/from the overlay network and to de-multiplex traffic from the overlay network to application-specific plug-ins. The key responsibilities of the Application Manager (AM) include logging QoS status information for each application and informing (when appropriate) the underlying overlay routing layer about the QoS status/requirements of the applications, and authorizing of a new user in the system based on a local policy. Each of these components focuses on different areas of functions with a common goal of providing best feasible QoS services to the applications and providing a layer of abstraction to the application developers.

Next, we describe in detail how these components interact to provide the required functions of the AWON.

4.1.1 Application-aware plug-ins

In the application-aware paradigm, each application is required to configure its functionality in the participating overlay nodes. AWON supports ASPs that implement the functions performed by the participating overlay nodes during the data dissemination. For a particular application, multiple nodes can play different roles, motivating the need to deploy relevant ASPs on those nodes that implement particular functions. For example, with a collaborative radar application [Mc05], a node identified as a source node in Fig. 4.1 has ASP 1 shown in Fig. 4.2 for supporting data selection and marking. Similarly nodes 1-5 in Fig. 4.1 may have ASP 2 to support application-aware forwarding based on the source marking. Nodes 4, 7, and 8 may have ASP 3 to support application-aware multicasting and congestion control. Note that the same node may have multiple ASPs to support multiple functions performed by a node for a given application. For an

example, in Fig. 4.1, node 4 acts as a forwarding node and a multicasting node for the same application. ASPs are dependent on AWON which provides services, including a way for communication by which data is exchanged, and a mechanism for requesting QoS requirements from the overlay network. For implementation of AWON based application-aware network, proposed API can be used by application programmers to create ASPs that access the services offered by the AWON. This API and the internal mechanisms of the AWON are described in Section 4.2.

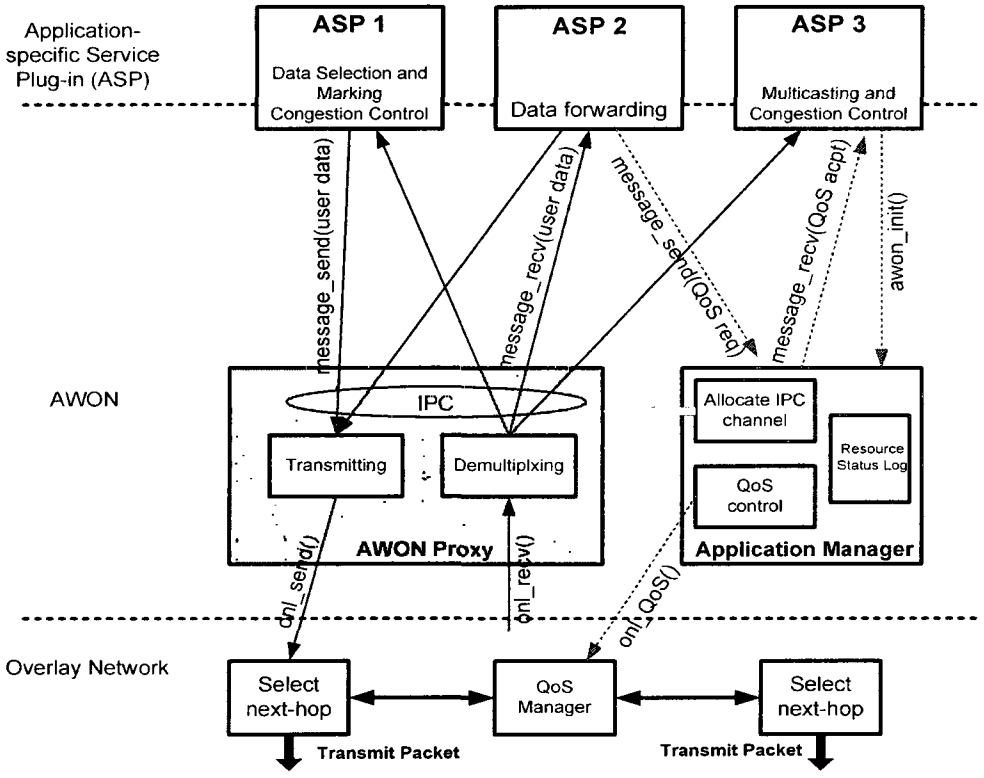


Figure 4.2 The AWON architectural framework

4.1.2 Interfacing with overlay networks

Since most of the overlay networks have their own send/receive API and specified packet format, AWON may need to provide overlay convergence functions for overlay networks to utilize the

packet delivery functions of the overlay networks [Jo06]. As illustrated in Fig. 4.2, the AWON Proxy (AP) is the interface between ASPs and overlay networks. The AP accepts items of data from the ASPs and encapsulates the data in the format specified for a particular overlay network for transmission. The AP is also responsible for receiving packets from the overlay networks. As data packets arrive at an overlay node, the AP de-multiplexes the packets to the proper ASPs for application-specific processing.

As seen in Fig. 4.2, AWON also offers a facility to the ASPs to specify their QoS requirements. At the initializing stage of an ASP, the ASP sends its QoS requirement to the Application Manager (AM), and the AM informs the overlay routing layer of an application's QoS requirements. The overlay routing protocol may refer to application's QoS requirement in their path selection for the application.

4.1.3 Concurrent usage of the overlay network

As illustrated in Fig.4.1, the same overlay node may have multiple ASPs to support multiple functions performed by a node for a single application as well as multiple applications. In order to deploy multiple ASPs on the same overlay node, AWON needs to provide a concurrency mechanism. Both threads and processes can be used to implement the concurrent ASPs. The cost, in both time and memory space, of creating and maintaining state information for each thread is much cheaper than the cost of a process. In addition, the thread-based ASPs can communicate with other modules (e.g., an ASP and the AP, the AP and the AM) in the framework directly because they can share the same memory space. However, it is difficult to add new ASPs to the overlay node, and update the functionality of ASPs without stopping the existing application-aware services in a thread-based implementation. Ever increasing diversity in application types and their QoS requirements may demand frequently changing to the functions of overlay nodes, so we need to provide a flexible way to change the functionality of ASPs. Therefore, in spite of

the advantages of the thread-based implementation, we selected independent process-based implementation for the ASPs. Using the process-based plug-ins implementation, we can deploy new ASPs without affecting other existing ASPs and other modules in the AWON, and delete an ASP when no longer required. However, the inherent separation between processes can require a major effort to communicate between ASPs and the modules of the AWON architecture, or to synchronize their actions. In order to deal with the problem, AWON implements an IPC (Inter-Process Communication) mechanism between ASPs and AWON, and presents an API which shields application programs from implementation details of the IPC mechanism; therefore, it is not necessary for the application developer to have any knowledge about the internal details of overlay networks and the framework. In addition, the deployment of too many ASPs or acceptance of large number of users above the resource availability of an overlay node may degrade the performance of all the ASPs which are running on the same overlay node. To avoid this situation, AM performs the book keeping of resource utilization and availability on the overlay node and authorize new user/ASP based on the local resource availability.

4.2 Application Programming Interface

One of the important tasks of AWON is to provide a simple way for user applications to access the services offered by AWON. The services include a way for ASPs to specify their QoS requirements and a communication mechanism by which ASPs send/receive data to/from overlay networks. In order to facilitate its use, AWON provides an API. There are three broad categories of the API: (i) API calls to register/deregister an ASP with the AWON, (ii) API calls for communication between ASPs and AWON, and (iii) API calls for communication with overlay networks. The first and second categories of the APIs allows ASPs to access services provided by the AWON, The third category of the APIs is used for interfacing with overlay networks and are

not exposed to the application-aware service plug-ins or application developers. However, we describe the third category of the APIs in this section as well as the first and second categories of the API in order to provide a figurative template for implementing convergence modules for various overlay networks.

4.2.1 API calls for initialization/termination of an ASP

As shown in Fig. 4.1 and Fig. 4.2, each participating node from source to the destination may play a different role for a particular application.

The *awon_init()* and *awon_terminate()* APIs are used to register/deregister a new ASP with the AWON for a particular application.

handle awon_init(app_id, pref): An ASP registers itself with AM and informs the AM of the QoS requirements of the application using *awon_init()* when the ASP is newly created. The application identifier *app_id, pref* are passed in the *awon_init()* call. For a given application, a unique application identification *app_id* is defined and is used as an input parameter. It is important to note that *app_id* is a globally unique identifier for a given application deployed over the overlay network. The *pref* field represents QoS preference of the application such as latency, bandwidth requirement. Upon receiving ASP's initialization request via the *awon_init()* call, the AM checks the status of resource usage by existing ASPs, and if the resource is available for the new ASP it passes the QoS preference to the overlay routing layer to be used in its path selection. At the same time, the AM allocates a new IPC channel for the new ASP in the AP, and return a *handle*. The handle contains incoming/outgoing IPC access points for data exchange between AWON and ASPs and *app_id*, and the handle is used to exchange data between ASPs and AWON as a parameter for the API call described in Section 4.2.2.

void awon_terminate(app_id, msg_buff): The *awon_terminate()* is used to tell the AM that the ASP will be terminated. Upon receiving ASP's terminating message via the *awon_terminate()* call, the AM releases the IPC channel and resources which are allocated to the ASP.

4.2.2 API calls for message exchange between ASPs and AWON

An API is required for three different types of messages that are exchanged between APSs and the AM as shown in Fig. 4.2:

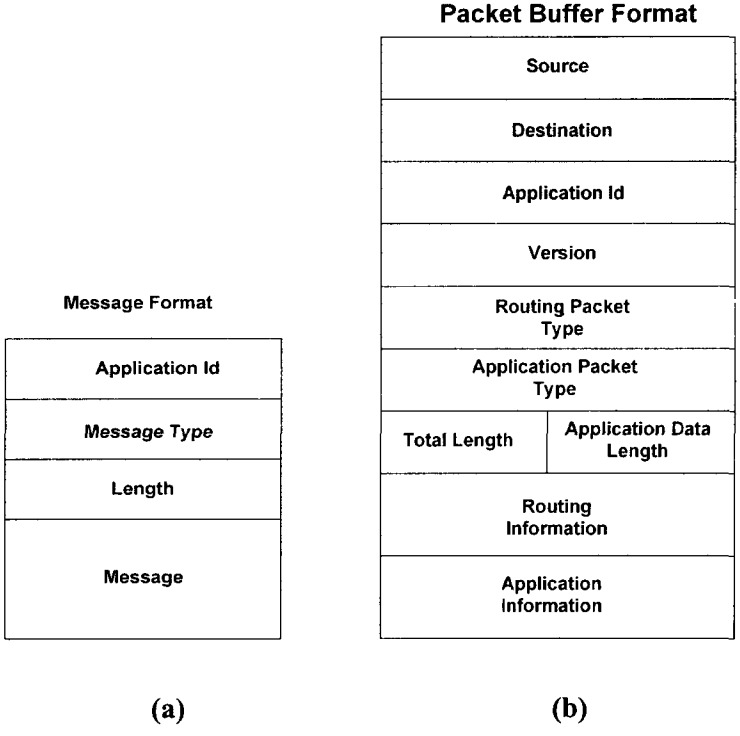


Figure 4.3(a) msg_buff format used for communication between application manager and application plug-ins. (b) pkt_buff format - an example

(i) User data exchanged with AWON to send/receive through the overlay routing layer for the user applications

(ii) Authorization messages to allow new users in the system

(iii) Periodic exchange of application-specific QoS messages

We define two API calls for message exchange between an application and the AWON, *message_send(), message_recv(), and recv_upcall()*. Fig. 4.3(a) shows a possible structure of the message format.

int message_send(handle, msg_buff): As shown in Fig. 4.2, the *message_send ()* API is used by the AWON and ASPs to send messages to each other within the same node. It accepts two input parameters, *handle* and *msg_buff*. The *handle* is a unique IPC access point identifier for a destination module in the AWON. *msg_buff* is the actual message sent to the destination node. It returns 1 when message is successfully transmitted otherwise it returns 0.

int message_recv(handle, msg_buff): As shown in Fig. 4.2, *message_recv()* is used by the ASP and the AM to receive messages from each other. It accepts two arguments, *handle* and *msg_buff*. The *handle* is the IPC access point of the message sender. *msg_buff* contains the copy of the message received from the sender side. Now we explain *msg_buff* format in detail as follows:

Application Id: This is the unique id of the ASP.

Message Type: There are two categories of messages that are exchanged between the ASP and the AWON depending on the context: (i) *UsedData*, and (ii) control message for exchange QoS information with the AWON. Control message types can be *QosRequest*, *QosAccep*t, or *QosStatus*.

Length: The *Length* field indicates the size of the *awon_msg* that includes variable length *message* field.

Message: The *message* field content varies with *message_type*. When the message type is *QosRequest,* the message field contains *target_bw, minimum_bw,* and *latency* requirement *fields*, and when the *message type* is *QosAccept* or *QosStatus*, message field stores TRUE or

FALSE flags. Alternatively, when the message type is *UserData*, the message field contains application data. It returns 1 when message is successfully transmitted otherwise it returns 0.

4.2.3 API calls for communication with overlay routing layer

An API is also required to support communication between the overlay routing layer and AWON. As seen in Fig. 4.2, AWON architecture requires the following communication support:

(i) Packet delivery from the AP to the overlay routing layer

(ii) Packet delivery from the overlay routing layer to the AP

(iii) Exchange of an application's QoS requirements with the overlay routing layer

To support these requirements, we define three APIs under this category, *onl_send(), onl_recv(), and onl_QoS()*.

int onl_send(dest, pkt_buff, length): The *onl_send()* function is used by the AP to transmit application data using the overlay routing protocol. It accepts three input parameters *dest, pkt_buff,* and *length*. *dest* refers to the destination address of the packet, *pkt_buff* is the pointer to the application packet buffer, and *length* indicates the size of the *pkt_buff*. It returns 1 when packet is transmitted successfully and returns 0 otherwise. The *onl_send()* selects the next hop for transmission based on the overlay routing protocol implementation. Alternatively, the application may use overlay source routing to route packets through pre-determined paths. Under most circumstances *pkt_buff* should contain *app_id*, *source*, and *destination* addresses. Fig. 4.3(b) shows a possible structure of the application packet, i.e., *pkt_buff*. All non-shaded fields are configurable and can be determined based on application-specific characteristics and in conjunction with the overlay routing protocol used in the network. The different packet fields shown in Fig. 4.3(b) are as follows:

Source: The unique address of the source node.

***Destination*:** The address of the remote sink node or the next hop in the path from source to the destination sink node.

Application Id: The unique application identifier which is set by the application plug-in module at the time of transmission.

Version: Current version of the packet format.

Application Packet Type: Depending on the application, this field is used to indicate the contents of the packet data. Some of the possible packet types are: *DATA, ACK, REQUEST, TERMINATE.*

Routing Packet Type: Depending on the routing protocol implementation, this field is used to define different packet types that can be used by the routing protocol to select the next hop for routing the application packet. Some of the possible routing packet types are *SOURCE_ROUTING, QOS_ROUTING.*

Total Length: The size of the packet in bytes.

Application Data Length: The size of the *Application Information* field in the packet in bytes. The value of this field may vary from application to application, and can also vary from packet to packet within the same application.

Routing Information: The contents of this field are defined based on the overlay routing protocol implementation. For example, when QoS-aware routing is supported, then an overlay routing protocol may store QoS requirements that should be considered for the next hop selection. When source routing is used, then this field includes the complete path to be followed between source and the destination node.

Application Information: The content of this field is determined by the application transmitting the data. It may contain application-specific header and data payload. Some of the possible application-defined fields are *sequence number*, *packet marking*, *QoS requirements*, and *metadata*.

int onl_recv(pkt_buffer) : As shown in Fig. 4.2, the *onl_recv()* API call is used to receive data from the overlay routing protocol layer. It accepts one argument, *pkt_buffer*, which is a pointer to the packet received from the overlay routing layer. Based on the *application id* field in the received packet, the packet is demultiplexed to the appropriate application. When a packet is received successfully, *onl_recv()* returns length of the packet otherwise -1 is returned.

void onl_QoS(app_id, bandwidth, latency): As shown in Fig. 4.2, the *onl_QoS()* API is called by the application manager to inform the overlay routing layer of an application's QoS requirements. It accepts three input parameters, *app_id*, *bandwidth*, and *latency*. *app_id* is the application id for which QoS requirements are specified, *bandwidth* is the minimum bandwidth requirement of a particular application, and *latency* is the maximum latency that an application may tolerate.

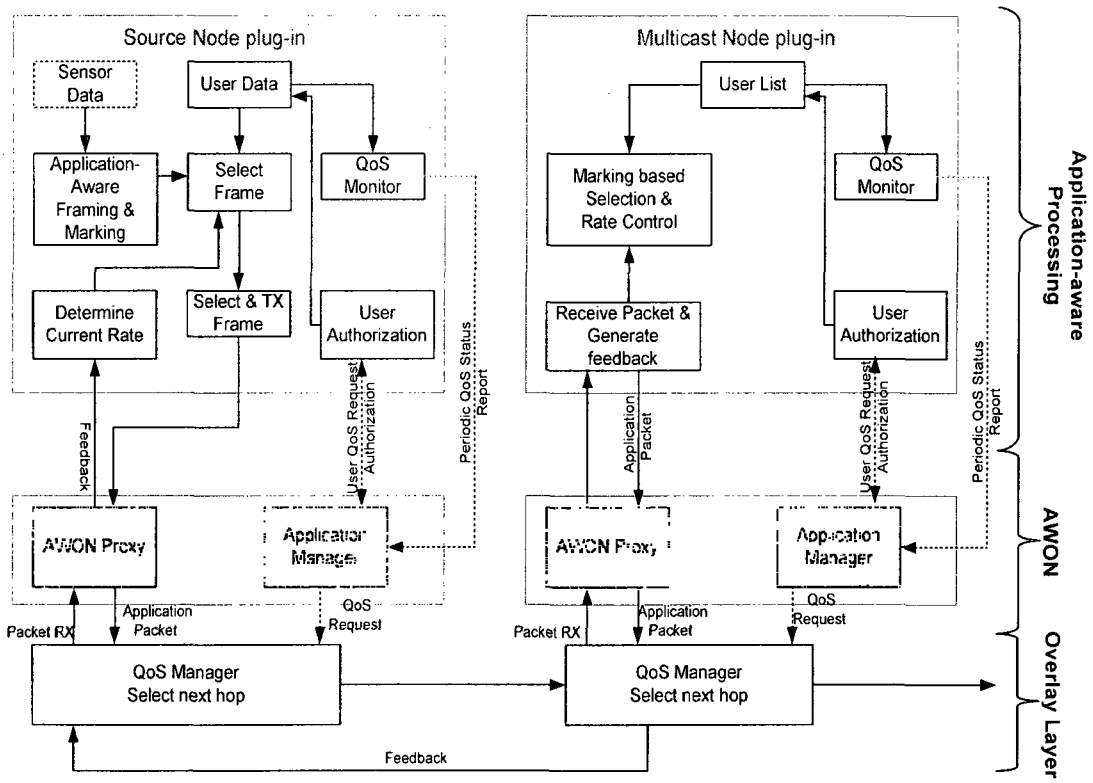


Figure 4.4 Implementation example based on AWON architecture

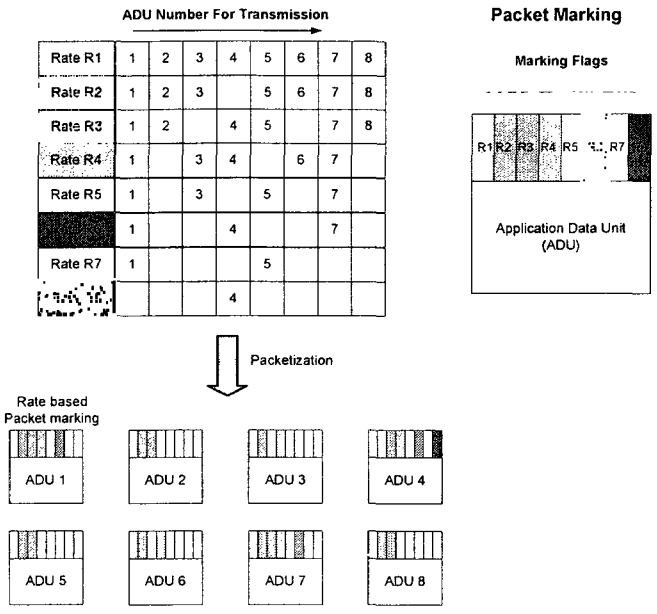


Figure 4.5 Application-aware framing and packet marking. Each non-white color represent rate for which packet is marked ,i.e., rate R1-R8 [Lee06].

4.3 An example Implementation: application-aware one-to-many data distribution.

To demonstrate AWON capabilities, we consider a CASA application as shown in Fig. 4.1, where data from a radar source node is distributed to multiple end-users with distinct bandwidth and data quality requirements. In this application, an application-aware multicast node receives data from the source node for further distribution to multiple end-users. AWON is used to perform application-aware processing at source nodes and multicast nodes to best meet the QoS requirements of multiple end-users. Fig. 4.4 shows the implementation details of a source node and a multicast node based on the AWON architecture. Both nodes use application-specific plug-ins to implement application-specific functionalities. The AWON implementation is same for all nodes in the overlay network. As shown in Fig.4.4, the source node plug-in implements application-level packet marking and a rate-based congestion control algorithm. Packet marking

determines the subset of the information that should be transmitted at a lower transmission rate for acceptable data quality at the receiver end. Fig. 4.5 explains the marking scheme used in the current implementation in [Lee06]. Consider an example as shown in Fig. 4.5, where a sensor node generates 8 application data units (ADU) within the bounded time at rate R1. The ADU is defined as a fundamental application data entity that can be used by an end-user algorithm for processing. Each row in Fig. 4.5 shows the subset of ADU that are selected for transmission at a lower transmission rate when a higher rate cannot be supported because of bandwidth constraints. The subset of data selected at lower rate depends on the end user quality requirements. For example, certain end users need uniformly spaced ADUs when only a subset of the data can be selected for transmission. Alternatively, other end users prefer a contiguous group of ADUs when bandwidth is constrained. Consider the case when the source node transmits data at rate R1, and as seen in the figure the data transmitted at lower rates is a subset of the data transmitted at rate R1 and ADUs are selected uniformly at lower rates. The packet containing ADU 1 is marked with different color flags corresponding to different rates, i.e., rates R1-R7. Similarly packet containing ADU 3 is marked with different colors corresponding to different rates, i.e., R1, R2, R4, and R5. As shown in the Fig. 4.5, every packet contains a flag for each rate for which it is transmitted indicated by different colors. Note that multiple flags can be set to indicate suitability of the packet for multiple transmission rates. The QoS monitoring component of the plug-in monitors the quality of the service received by the application users at a source node. Currently, the component monitors whether end users' bandwidth requirements are met. The multicast application plug-in supports application-aware rate control using a token-bucket scheme and on-the-fly forwarding of data based on the packet marking. In Fig 4.6, packet-marking algorithm primarily consists of two key phases: (i) Processing of Arriving Packets, and (ii). Processing Packets for Transmission. As seen in the figure, at the beginning of every heart-beat interval, algorithm determines the token bucket size N[END_USER] for the END_USER based on its current transmission rate. The heart-beat interval is divided into S time slots of equal duration.

Let B[END_USER,i] denote the number of tokens allocated to the END_USER in the time slot i. Therefore, N[END_USER]/S tokens are evenly allocated to each time slot, i.e., B[END_USER,i]=N[END_USER]/S. When a new packet arrives, it is pushed to the appropriate end user queue. At the time of transmission, for each time slot i algorithm checks for the number of tokens present in the bucket for each end user. If the tokens are present in the bucket N[END_USER] and corresponding queue is not empty, then the packet is popped from the queue and transmitted to the END_USER, and B[END_USER,i] and N[END_USER] are decremented by 1. At the end of time slot i, if B[END_USER,i] is greater than 0, it is the indication of missing packets with desired marking for a particular END_USER. Since data may also arrive in bursts, therefore B[END_USER,i] can be less than 0. For e.g., if B[END_USER,i]=2 and then in the ith time slot 3 packets with required marking arrives then B[END_USER,i] is -1 at the end of ith slot. At the end of the ith slot, B[END_USER,i] is subtracted from the L[END_USER] to track the difference between total expected packets and the actual received packets until the end of the ith time slot. L[END_USER] >0 indicates that more number of packets with the desired marking arrives than the expected number of packets at the overlay node until that instant. Alternatively, L[END_USER]<0 indicates that missing or delayed packets with the desired marking. When L[END_USER]<0 and is below an acceptable threshold value THRESHOLD then the protocol initiates marked packet compensation process in the (i+1)th time slot. During the packet compensation process, multicast node also start selecting packets with marking corresponding to rates higher than the current transmission rate R[END_USER] of the END_USER. Therefore queue for END_USER may contain packets with marking corresponding to rates greater than R[END_USER]. This process continues as long as L[END_USER]<0 or until the expiration of the heart-beat interval. At the end of the heart-beat interval, there may be some marked packets left in the end user queue that couldn't be transmitted within heart-beat interval. In that case all queues are cleared leading to drop of marked packets at the overlay node.

```
                    Packet-Marking Algorithm
#heart_beat:  Periodic time interval to generate block of sensor data
#S:           Total number of time slots within heart-beat interval
#i:           Index of the time slot within heat-beat interval
#PACKET:      Packet received at the node
#PKT_SIZE:    Size of the PACKET
THRESHOLD:        Application defined threshold to initiate packet compensation
#R[END_USER]:     Current transmission rate of END_USER
#N[END_USER]:     Number of tokens in the bucket for END_USER
#B[END_USER,i]:   Number of tokens assigned to  time slot i
#L[END_USER]:     Number of missing marked packets for END_USER
#get_transmission_rate(END_USER): Determine current transmission rate for END_USER
#push(PACKET,END_USER):          Push PACKET on a queue for END_USER
#pop_and_transmit(END_USER):     Pop packet from queue for END_USER and transmit it
#clear(END_USER):                Clear queues for END_USER

#Initialization phase every heart-beat interval
FOR each END_USER {
   R[END_USER] = get_transmission_rate(END_USER) #Determine transmission rate for each end
user
   N[END_USER] = (R[END_USER]*heart_beat)/PKT_SIZE  #Determine token bucket size for each end
user
   FOR each time slot i=0; i <S;i++   #Assign token to each slot of heat-beat interval for END_USER
     B[END_USER,i] = N[END_USER]/S
     L[END_USER]=0   #Initialize missing packets to 0 for each END_USER
  }
#Process New Arriving Packets
FOR each arriving PACKET
   FOR each END_USER {
      #Check if the desired packet arrives based on packet marking
      IF (PACKET MARKING==END_USER MARKING) {
         push(PACKET, END_USER) #Push packet to queue for the end user
         }
      #Check if packet with higher marking than the desired arrives
      ELSE IF (PACKET MARKING > END_USER MARKING) {
         IF (L[END_USER]<THRESHOLD) { #Check if compensation phase
            push(PACKET, END_USER)  #Push packet to the queue for the end user
            }
         }
      }
 #Process Packet Transmission
  FOR each time slot i=0;i < S;i++ {
     FOR each END_USER { # Repeat following loop multiple time until current time slot expires
        IF  (N[END_USER] >0) {  # Check for presence of tokens in the bucket for end user
           IF  (!queue_empty(END_USER)) {
              pop_and_transmit(END_USER)  #Pop queue and transmit
              B[END_USER, i)  =  B[END_USER,i] -1  # Track number of packets received in time slot i

              N[END_USER]     = N[END_USER]-1  # Number of tokens left in the bucket for
END_USER
            }
         }
      }
   # After time slot i expires, update number of missing or excess packets until time slot i
   L[END_USER]     =  L[END_USER] - B[END_USER,i]
   }
  #Clear all queues at the end of heat-beat interval
  FOR each  END_USER
       clear(END_USER)
```

Figure 4.6 Packet-marking algorithm [Lee06]

The motivation for dropping remaining packets in the queues after the heart-beat interval is that these packets are likely to arrive late at the end user destination; therefore they may be useless for the end user application. It is important to note that performance of the token bucket based rate control is sensitive to the duration of the heart-beat interval. Token bucket based algorithm will be able to select the most appropriate marked packets for forwarding if delay jitter is small compared to the heat-beat interval. The multicast node ASP selects data for forwarding based on the available network bandwidth and the packet marking for multiple end users. Note that the packet marking performed at the sender node determines the priority of the packet to be forwarded at the multicast node. In such systems, each end user may need a different subset of the data from the radar source based on the intended use of the data. During network congestion, overlay nodes can perform a better job by selectively dropping packets (information) instead of dropping randomly within the network, taking into account end-user requirements for different subsets of the data. Similarly, other nodes participating in the data transmission can play different roles and thus may have different plug-in implementations. For example, if a node is configured as a simple forwarding node for a particular application, then the application plug-in forwards the data to its overlay routing layer for further processing. The overlay routing layer may then select the next hop for the transmitted packet based on the local policy. If QoS-aware routing is supported, the next hop may be selected based on the application's QoS requirements. Alternatively, when source routing is used, the routing protocol would select the next hop based on the path information included in the packet.

4.4 Experimental Results

The real benefit of the AWON is to facilitate the deployment of application-aware service on overlay networks. However, in order to widely deploy in the overlay networks, the additional

latency incurred by the AWON has to be low. We first demonstrate that the overhead of the AWON implementation is not large by showing the latency for packet sending/receiving through the AWON. We also demonstrate the effectiveness of AWON architecture and the API for implementing and deploying real-time applications on planetlab test-bed [Pla].

4.4.1 AWON overhead

In comparison to common overlay network packet delivery that is not using application-aware processing at the intermediate nodes, the AWON architecture takes additional overhead to exchange data packets with ASPs to process the packets in an application-specific manner. In this first set of experiments, we determine the latency overhead imposed by AWON. Test programs that send/receive packets at a specified rate played the role of application-aware plug-ins. The source node plug-in generates packets at about 1Mbps, and the intermediate node plug-in simply forwards the packets to the receiver node plug-in without any in-network processing. Thus, the overhead of the AWON architecture is localized from application-specific in-network processing time. The experiments were conducted on 3.06 GHz Xeon machines with 2 GB RAM running Linux 2.6.18, and each timing statistic reported here is a 100 sample mean of 10000 packets. The latency due to the AWON architecture is split into four phases within a single overlay node: (a) time to move a packet from the overlay network to the AWON proxy for receiving, (b) time to move a message from the AWON proxy to an ASP, (c) time to move a message from the ASP to the AWON proxy for sending, and (d) time to move a packet from the AWON proxy to the overlay network for sending. Fig. 4.7 shows the send/receive latencies of a single packet of size 1400 with varying the number of plug-ins in a single node. As seen in the figure, most of the overhead is attributable to the IPC between the AP and ASPs, e.g., more than 93% of the total time is spent for the communication with ASPs for the 5 ASP case. The remaining overhead is due to the data exchange with the overlay network. In order to transmit a packet through the

overlay network, the AP encapsulates the data in the format specified for the overlay network, so a data copy is required for the encapsulation. At the receiver side, the AWON extracts a packet from the overlay network, and the extracted packet is de-multiplexed to an appropriate ASP for application-specific processing. Before forwarding the packet to a particular ASP, the AP needs to copy the packet to convert from overlay-specific packet format to the AWON message format illustrated in Fig. 4.3(a). Thus, the overhead between the AP and the overlay network are mainly due to the memory copies. In Fig. 4.7 for the case with 9 ASPs, the average total overhead is 50μs, for example, in case of 10 intermediate nodes total 0.5msec additional overhead is imposed on the communication path from the source node to the destination excluding ASP's packet processing time at the intermediate nodes. In our testing environment a round time delay form a source node to a destination (a node in Massachusetts to a Colorado State University node) is about 80msec. Therefore, the low overhead imposed on each packet while going through AWON does not have any significant effect on the total latency from the source to the destination. The benefits gained from AWON far outweigh this small cost.

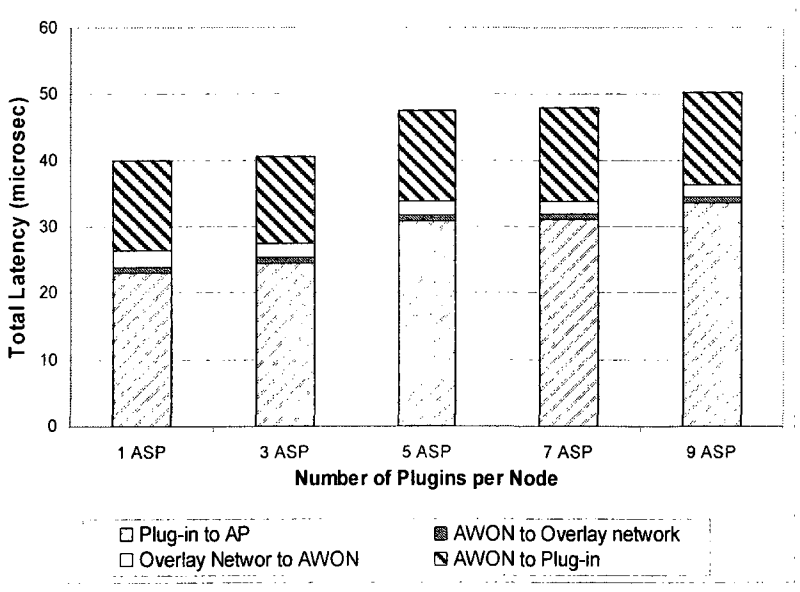


Figure 4.7 The per-packet latency for send and receive

4.4.2 Application-aware one-to-many data dissemination Application

We consider a mission-critical CASA application for the performance evaluation. One of the requirements of CASA applications is to distribute high bandwidth real-time weather radar data to multiple end users with distinct critical bandwidth and data quality needs. For such applications, it is not only important to meet the bandwidth and latency requirement, it is also important to meet the minimum content-quality requirement for the proper operation of the system.

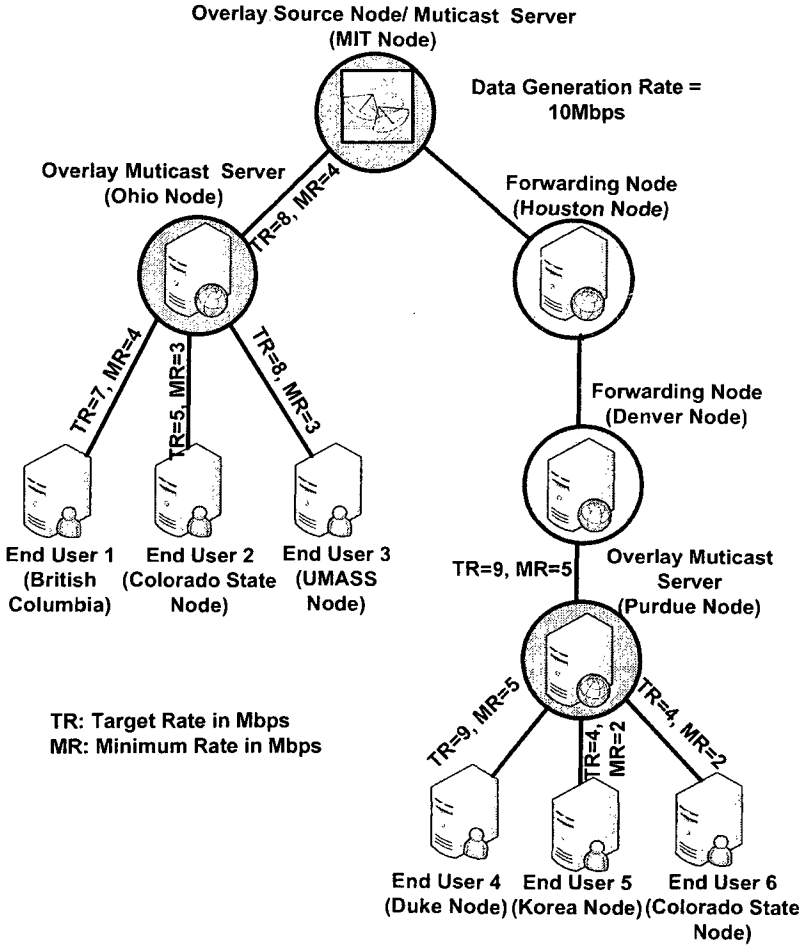


Figure 4.8 Planetlab test-bed for application-aware multicasting

For example, each CASA end user may specify its critical minimum rate (MR) requirement that should be met for the proper operation of the system. Moreover, each end user may also dictate a target rate (TR), i.e., the maximum rate at which data can be received by the end user. A source node periodically generates a block of digitized radar data, referred to as a DRS block [Ba05]. Each end user specifies its content-quality requirement in terms of tolerance towards bursty losses and/or uniform losses within the DRS block. In the current implementation, we consider a case in which all end users prefer random drops of information instead of bursty drops within a DRS block in case packets have to be dropped. In case of our CASA application, during network congestion, the desired rates are between MR and TR and the desired packets are those that contain subset of the DRS block of data with uniform drops. All these selected packets are marked for rate between MR and TR at the source node. We implement this application using the AWON architecture, as it enables application-aware processing within overlay nodes to enhance the QoS under dynamic resource-constrained conditions.

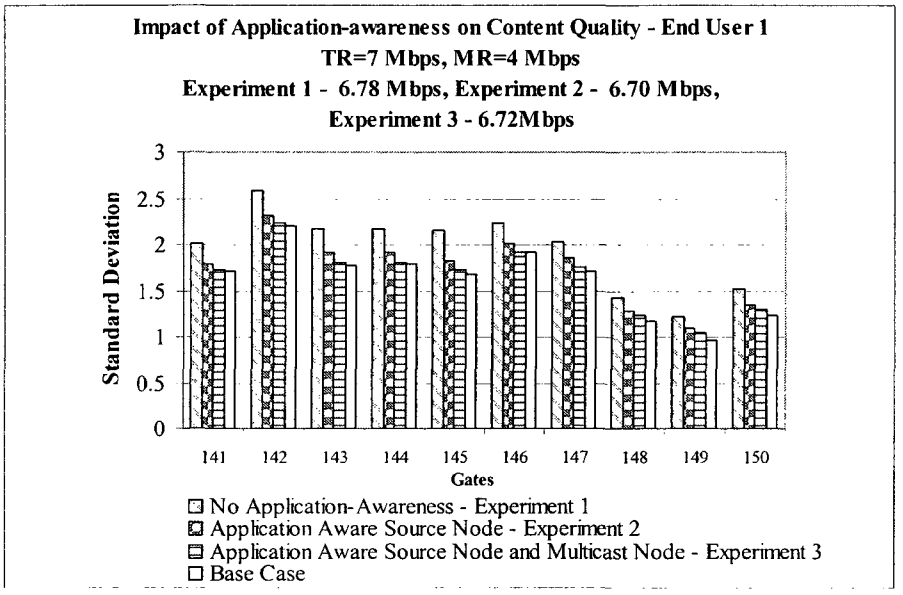
Overlay Network Topology: Fig. 4.8 shows the Planetlab based overlay network topology used for application-aware data distribution and performance evaluation. It consists of 11 overlay nodes, each configured to perform application-specific tasks to meet the overall QoS requirements of the application. In Fig. 4.8, there are four different types of nodes that are present in the overlay network - a *source node*, a *multicast node*, a *forwarding node*, and an *end user node*. The *source node* performs selective data drop during network congestion as well as application-aware packet marking based on the end user's data quality requirement as explained. The goal of the marking scheme is to deliver the most appropriate subset of data for the end user under congested network conditions. The *forwarding node* may decide to forward a packet based on a packet's marking and the available downstream link bandwidth. The *multicast node* performs on- the-fly selection of the data for forwarding based on packet marking to the respective end users at the current transmission rate. The *multicast node* uses TRABOL (TCP-Friendly Rate Adaptation Based On Losses) to independently determine the transmission rate for

each end user. The *end-user node* performs content quality evaluation using application-specific performance metrics and provides periodic feedback to the *multicast node* about its current receive rate. In Fig. 4.8, six different *end-user nodes* 1-6 at geographically different locations receive weather radar data streams from the source node at *MIT* at their required TR and MR over the Planetlab.

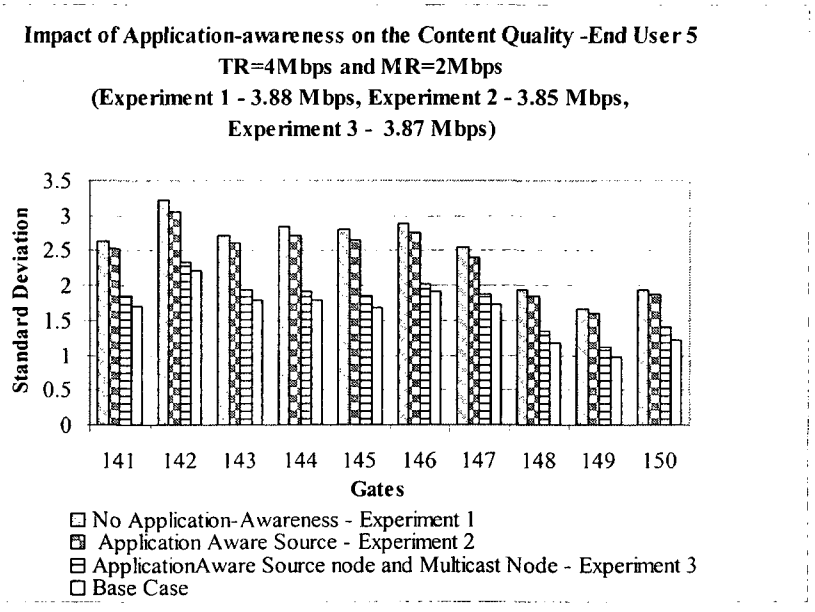
Performance Metrics: The effectiveness of the AWON architecture and the programming interface can be evaluated by measuring the quality of the content delivered to the end users under different network congestion conditions. For most real-time applications, application-specific metrics are used to measure quality of the content; for multimedia applications, these metrics include PESQ for voice quality and PSNR for video streaming. For the CASA application we use the standard deviation of the estimated sensed values (specifically, reflectivity and wind velocity) to evaluate quality of the radar data [Ba05]. A lower standard deviation indicates better radar data quality. A minimum standard deviation, i.e., the highest content quality, is achieved when all the data from the source node is delivered to the end users. Alternatively, we also evaluate the content quality by measuring the frequency of the desired packets at the receiver node based on their markings. For better quality of the data, it is necessary to receive more packets with the desired markings. For an application with TR and MR bandwidth requirements, the "most appropriate" packets are marked to result in data rates between MR and TR.

Methodology: We demonstrate the effectiveness of the AWON architecture for application-aware processing within the overlay network by performing three sets of experiments. In the first set of experiments, experiment 1, no application-aware processing is performed in the network, i.e., the source node randomly selects data from a DRS block of radar data for transmission, without considering end-user loss tolerance requirements. Packet marking is performed but packet marks are not used at the forwarding nodes or at the multicast nodes for on-the-fly selection of packets for transmission. In experiment 2, the source node performs application-aware selective drop during network congestion and marks packets at the time of transmission.

However, packet marking is not used at forwarding nodes and multicast nodes for on-the-fly selection of data for transmission to the end users. Experiment 2 is equivalent to a network that supports limited application-aware processing at end hosts without the support of AWON architecture. Experiment 3 is an example of the AWON-based implementation that enables in-network processing by performing different application-specific tasks within the network. In Experiment 3, the source node at *MIT* performs application-aware selective drops and packet marking. The multicast nodes at *Ohio* and *Purdue* use token-bucket based rate control scheme along with packet marking to select appropriate packets on-the-fly for transmission to individual end users at their respective transmission rate. At present, in experiment 3, nodes at *Houston* and *Denver* act as simple forwarding nodes and do not make use of packet marking when forwarding packets. Performance is compared by measuring the quality of the content delivered to the end users for different experiment scenarios under different network congestion conditions. For lack of space we show results for two end users, End user 1 and End user 5. As mentioned earlier, data is generated at 10Mbps at the source node but end user 1 requests for TR=7Mbps and MR=4Mbps. End user 5 has relatively lower bandwidth requirement with TR=4Mbps and MR=2Mbps. Both end users can tolerate uniform drop of data within the DRS block. Both end users compute reflectivity using raw data received from the radar source node. Fig. 4.9(a) and 4.9(b) show the standard deviation of reflectivity for all three experiments. In this radar application, each end user computes reflectivity for multiple gates [Ba05]. (In radar terminology, a gate refers to a volume in the atmosphere at a particular distance from the radar source node for which data is collected by a radar.) Fig. 4.9 thus shows content quality, i.e., standard deviation for subset of gates. As seen in Fig. 4.9(a) and 4.9(b), experiment 1, with no application-aware processing support within the network, has highest standard deviation and hence has the worst data quality among three cases. In experiment 2, when limited application-aware drops are performed at the source node, the quality of the data improves in comparison to experiment 1, as indicated by decrease in standard deviation.



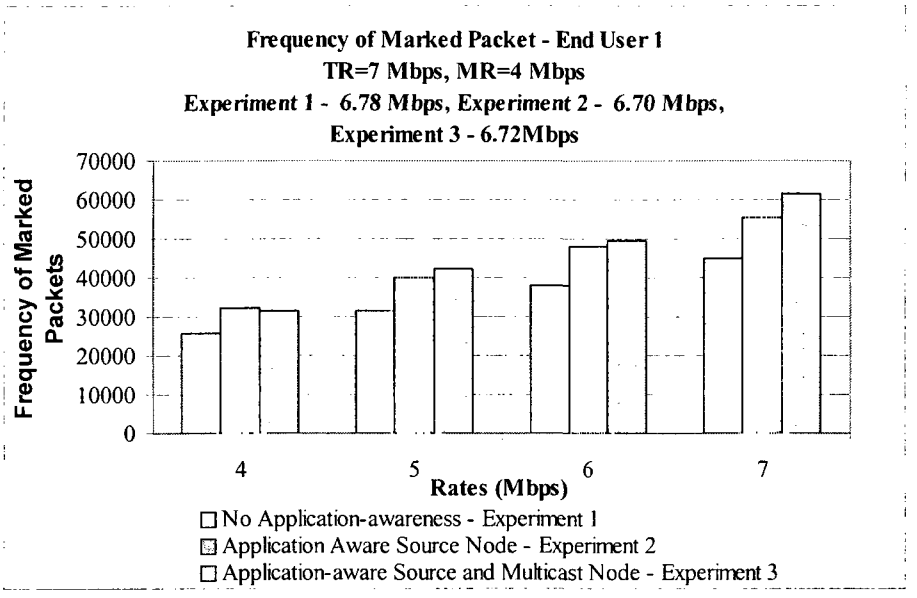
(a)



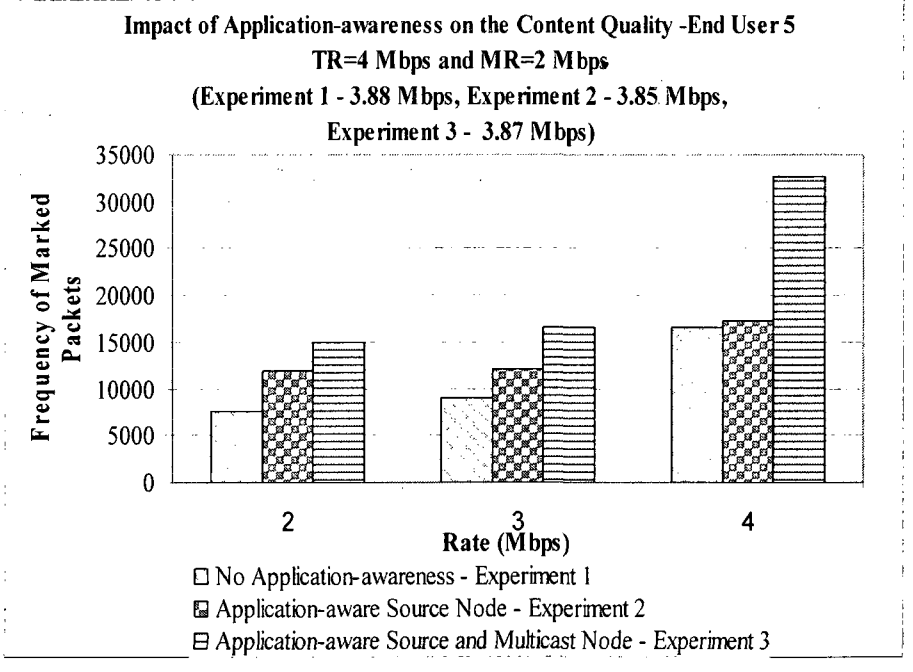
(b)

Figure 4.9 Impact of application-aware architecture on the content quality delivered to the end user s (a) Standard deviation of data for end user 5 with low bandwidth requirement TR=4, MR=2, (b) Standard deviation of data for end user 1 with high bandwidth requirement [Ba07]

Experiment 3, which has support for application-aware drop at the source node and marking-based selective drop at the multicast nodes, delivers data with the highest quality, i.e., with the smallest standard deviation. It is important to note that under high loss conditions, the AWON architecture is very effective in improving the quality of the data as shown in Fig. 4.9(b). Indeed, the standard deviation of the AWON case approaches that of the *base case* standard deviation, which corresponds to a scenario when *all* data from the source node generated at 10Mbps is For example, in Fig. 4.9(b), end user 1 receives data at 3.88Mbps, 3.85Mbps, and 3.87Mbps for experiment 1, 2 and 3 respectively. However, the application-level quality of data delivered to the end users is significantly different for all gates. The gain in performance in terms of content quality is achieved because AWON modules deliver the most appropriate application-specific content to the end user within the available bandwidth resources. This is made possible by performing application-aware processing of the data as it traverses the network. Fig. 4.10(a) and 4.10(b) show the impact of the three experiment scenarios on the delivery of most appropriate information to the end user at a given rate. Packets are marked for different rates for which it is most suitable for transmission as explained in Section 5. When an end user receives more packets with markings corresponding to the desired rate, this is an indication of a higher quality of received data. Aforementioned, for CASA end users, the desired rates are between MR and TR and the desired packets are those that are marked for rates between TR and MR. In Fig. 4.10(a) and 4.10(b), we show the number of packets delivered with the marking corresponding to rates between TR and MR requirements of the end users. Fig. 4.10(a) and 4.10(b) both measure content quality using different metrics and corresponds to the same end user 1. Fig. 4.9(b) and 4.10(b) illustrate the content quality for end user 5. As seen in the Fig. 4.10(a), and 4.10(b), experiment 1 with no application-awareness, delivers fewer packets with the desired marking. Alternatively, the frequency of the packets with desired marking increases with experiment 2 resulting in a higher content quality.



(a)



(b)

Figure 4.10 Marked packet frequency for end user 5(a), Marked packet frequency for end user 1 (b) [Ba07]

In the case of experiment 3, the frequency of desired marked packets is the maximum over all three cases. As seen in Fig. 4.10(b), during high network congestion, AWON based architecture is able to deliver 50% more desired packets than the case when no application-aware processing is done in the network. These results corroborate the results shown for data quality in Fig. 4.9(a) and 4.9(b), which used the standard deviation quality metric for end user 1 and end user 5 respectively. The above experiments demonstrate that the AWON architecture enables the deployment of application-aware services in the overlay networks and that such overlay services can be very effective in improving the performance of an application in resource-constrained conditions.

4.5 Summary

An Application Aware Overlay Network (AWON) architecture and a programming interface for the application-aware data dissemination has been proposed and implemented. The test results show that the AWON architecture provides many benefits to the application while keeping additional overhead at low acceptable levels. In addition, the Planetlab based experiments demonstrated the suitability of the AWON architecture and the programming interface for the deployment of application-aware services in overlay networks. We have seen that under resource-constrained conditions and/or network congestion, an AWON-based data dissemination application can deliver better quality data to the end users than a data-quality-oblivious implementation while using a similar amount bandwidth. The AWON architecture and programming interfaces are generic and are not limited to a particular application. It can thus be used to deploy applications that need application-specific processing within the network to meet its QoS requirements.

Chapter 5

AN ARCHITECTURAL FRAMEWORK FOR THE REAL-TIME IMPLEMENTATION OF CASA MULTI-RADAR DATA FUSION

In the CASA radar network, the radars collaborate to sense and detect weather events by focusing their scans on overlapping region in the atmosphere. It is often beneficial to simultaneously observe the same events through the use of multiple radars in different locations [Ku06] as the observations from multiple radars offer different, but complementary views of the same weather features. In this multi-radar collaboration environment, multi-radar data fusion application is increasingly common and becomes an essential constituent of the systems. The multi-radar data fusion necessarily involves the collection of data from multiple remote sources and the computation of intensive digital signal processing, and such properties of the applications leave unique demands on system infrastructure and application developers. In this chapter we present an architectural framework for multi-radar data fusion. Using the framework we seek to offer efficient system support for multi-radar data fusion applications. In order to support the multi-sensor data fusion applications, the framework needs to (i) provide an easy to use mechanism that simplifies multi-sensor fusion tasks; (ii) offer an efficient sensor data managing and searching mechanism for finding a set of data to be integrated, and (iii) provide a multi-process coordinate mechanism to exploit parallel/distributed processing for the computing intensive multi-radar data fusion.

The framework consists of a sensor database that contains a list of radars and related information such as their locations and radar measurements, a data management module, a common volume finding module, and a fusion algorithm processing module. Acquiring the desired accuracy from the final results of the algorithm requires the use of a set of data items to be integrated with the consideration of generation time gaps and the physical distances of data capture points among the data items. Because each of the radars can be assigned to a certain sensing task with a different scanning interval and mode in the CASA radar network, it is very difficult to synchronize the data generation time and the number of radar sweeps in a data generation interval. Furthermore, each of the radars generates tens of mega bytes data during a single data generation time interval. Due to the complexity of the radar data generation pattern and the large amount of data, managing data and finding a set of correlated data items for multi-sensor data fusion applications are considerable tasks. Taking all these aspects into account, we design and implement a sensor database, a data management module and a common volume finding module that provide efficient data managing and searching mechanisms for multi-radar data fusion applications. Furthermore, radar data fusion algorithms require substantial computation for intensive digital signal processing and have a strict real-time processing requirement. In order to realize real-time performance, hardware-based solutions with dedicated architecture design or special purpose programmable digital signal processors (DSPs) can be used for the multi-radar data fusion algorithms. However, the hardware-based solutions are less flexible and unsuitable for constantly evolving radar algorithms. In contrast, software solutions using general-purpose computing platforms are more flexible, and allow optimal algorithm updates from time to time, so there are plenty of radar algorithm developments based on the general-purpose computing platforms. In addition, general purpose computing platforms are getting more powerful and cheaper every year. Recently, symmetric multiprocessing (SMP) where two or more identical processors have equal access to a single shared main memory and chip-level multiprocessing (CMP) where one CPU chip contains two or more complete and independent functional units (cores) are proliferating to exploit parallel processing in a cost effective manner. To

make the most of a multi-core processor or multiprocessor system today, the multi-sensor radar fusion processing algorithms operating on the general-purpose computing platform must be implemented such that they can spread their workload across multiple execution cores or processors to achieve a high processing speed. Our framework is implemented in a multiprocessor environment, where multiple processes operate simultaneously and collaboratively on multiple processors or cores in a system to meet the real time requirement of CASA. This framework is used for the real-time implementation of a multi-radar data fusion algorithm in CASA-IP1, the first test-bed of the Center for Collaborative Adaptive Sensing of the Atmosphere. The effectiveness of the proposed framework is demonstrated for the data fusion algorithm, the CASA network-based reflectivity retrieval algorithm.

The rest of this chapter is structured as follows. Section 5.1 introduces the overall architecture of our multi-radar data fusion framework. Section 5.2 describes sensor database and data management module. In Section 5.3 we describe the common volume finding module. We describe the algorithm processing module that is implemented based on multithreading in Section 5.4. In Section 5.5 we present empirical measurements of the execution runtimes of the program. We summarize this chapter in Section 5.6.

5.1 Architectural Framework Overview

The DCAS applications often implement a closed-loop control system to provide the functionality required for data collection, computation, and actuation. In CASA, the loop begins with continuous data capture using radars, followed by ingesting data from remote radars, identifying meteorological features in this data, and ends with determining each radar's future scan strategy based on detected features and end-user requirements [Zi05]. An iteration of the

control-loop has to be done in a certain time period known as the system heartbeat interval. This heartbeat interval is selected based on the physical and mechanical properties of radars, the timescale over which atmospheric conditions change, and the system goal of detecting and tracking tornados within 60 seconds [Zi05]. The CASA radars can be dynamically configured in terms of the set of sectors to be scanned during a heart-beat interval. MC&C (Meteorological Command and Control) [Zi05] [Ku06] of CASA takes end user requirements and weather conditions into account to determine the optimal radar configuration for a particular heart-beat interval. Each mechanically steerable radar has two degrees of freedom that enable control over the orientation and the altitude where the radar points and senses data [Li07]. In PPI (Plan Position Indicator) scan mode, the radar scans the atmosphere by first holding its elevation angle (attitude) constant and then changing its azimuth angle (orientation). Based on the radar configuration, the CASA radars generate data in several sequential elevation angles combining a surveillance scan (radar rotates through 360 degrees in the azimuth angle) and sector scans (radar rotates through less than 360 degrees in the azimuth angle) during a heart-beat interval. Typically, each of the radars collects data as follows: it emits a coherent train of microwave pulses and coherently processes reflected pulses fixing its elevation angle and azimuth angle. A radar then moves to the next azimuth and the process is repeated [Kr97]. After collecting data for a particular elevation angle it increments the elevation angle and collects data for another elevation angle. The data collected at a particular pointing angle (azimuth and elevation) is referred to as a ray, and the collection of all the rays for a single fixed elevation angle is defined as a sweep [Ch02]. After a series of sweeps, the antenna is lowered to the base elevation, which concludes the collection of one radar volume. In our framework, a ray is treated as an atomic unit that can be accessed and processed individually.

Fig 5.1 shows the data flow from the CASA radars to the final stages of multi-radar data fusion processing. In the CASA network, multiple radar sensors take the measurements of physical variables of the atmosphere, such as the reflectivity and wind-velocity.

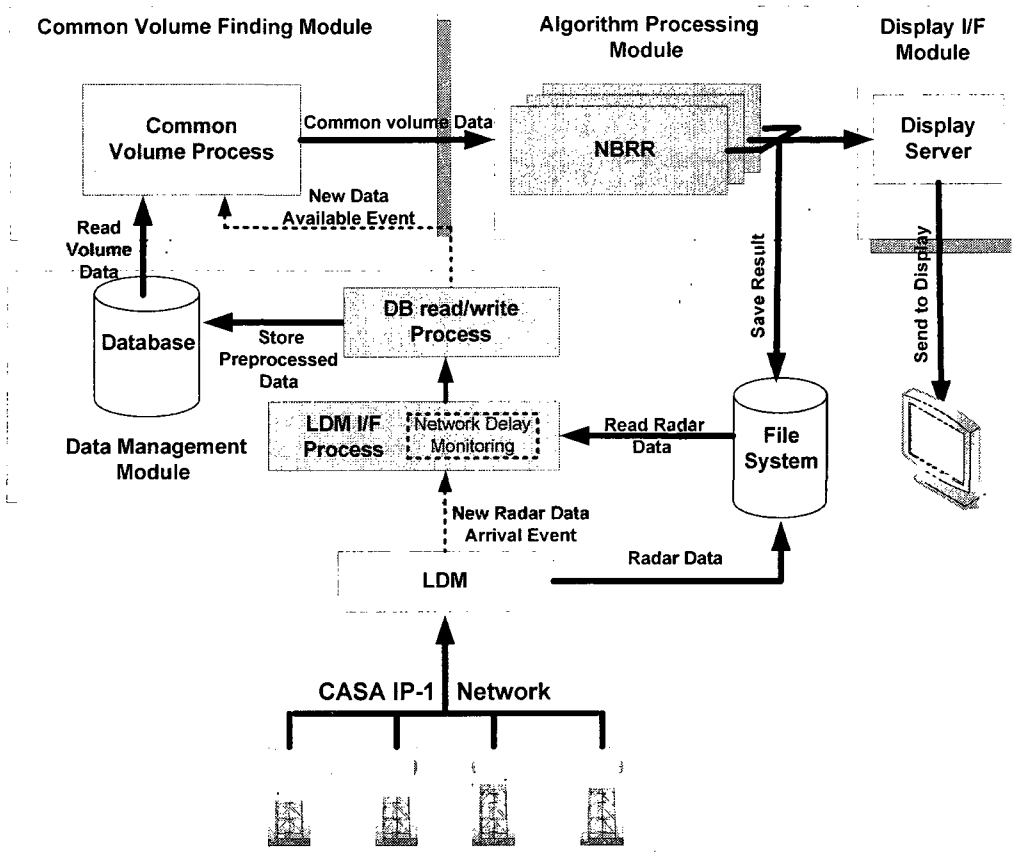


Figure 5.1 Multi-radar data fusion software architecture

The radar measurements are stored in a common file format, called netCDF and streamed from the radars in CASA IP-1 test-bed to the machine running the radar data processing algorithm. We use Local Data Manager (LDM) (www.unidata.ucar.edu/software/ldm/) as the communication software to obtain real-time data from the radars. The LDM posts the notifications of the availability of new data to the software whenever new data arrives from the remote radars. Once the notification has been posted, the database management module of the software reads the radar data and performs pre-processing including removing noise from the data and extracting radar operational parameters, and stores the data in the sensor database. The database management module informs the common volume finding module of updating the database after storing the

new data in the sensor database. In addition, the database management module removes aged data from the sensor database to maintain the recent volume of data from individual radars. The module also provides a data searching mechanism for the common volume finding module. Due to the need to merge data, it is required to select a set of correlated data. CASA multi-radar fusion algorithms require a set of spatially and temporally correlated data. We define a common volume as a set of data collected from an overlapping coverage area in the atmosphere at the same time interval by multiple radars. The common volume finding module uses the database searching mechanism provided by the data management module to find a common volume from the sensor database. The common volume finding module considers the physical distance as well as the time difference among the data to be used by the algorithm when it selects a common volume. The selected common volume is delivered via a message queue to the algorithm-processing module. The network-based reflectivity algorithm has several computing intensive loops to solve a set of integral equations. In the algorithm, each ray from the same sweep can be processed independently and individually. We implement the algorithm-processing module using multiple processes to aggregate computation resources for this most computing intensive part of the software. The algorithm-processing module runs multiple processes simultaneously to process multiple rays at the same time, so it reduces the execution time of the algorithm. Finally, the software writes the end results to netCDF files, and sends a real-time display server the final results to allow remotely located users to display the output of the software using CSU VCHILL display client [Ch05].

The software has been implemented using GCC with the GNU Scientific Library (GSL) (www.gnu.org/software/gsl/). In addition, we use Berkeley DB 4.6 to provide an access method to a disk based structure used to store radar measurements, radar parameters and data point coordinates and operations available on that structure.

5.2 Sensor Database and Data Management Module

Because the multi-radar fusion needs to use a proper set of data generated within the same time interval and captured at the same location in the atmosphere, it is required to identify data capture time and location. All the radars in the CASA IP1 network have GPS synchronized time, and each netCDF file contains data capture timestamp of each ray. Thus data capture time is easily identified using the timestamp in the netCDF file. Similarly, each data capture point in the atmosphere can be identified by a spherical coordinate system with three degrees of freedom (elevation φ, azimuth θ, and gate γ). A gate γ refers to a volume in the atmosphere at a particular distance from a radar. The common volume finding module requires the use of trigonometric equations to find a set of spatially correlated data from multiple radars. To ease the mathematical calculation using the trigonometric equations, the data management module converts radars' spherical coordinate system to a three dimensional Cartesian coordinate system centered at the mid-point of four CASA IP-1 radars. In the Cartesian coordinate system, the y-axis points north, the x-axis points east, and the z-axis represents the altitude, respectively. The coordinate system conversion is performed using the following equations:

$$
\begin{aligned}
x &= \gamma \cos[\phi \cdot (\pi/180.0)] \cdot \sin[\theta \cdot (\pi/180.0)] + \Delta x, \\
y &= \gamma \cos[\phi \cdot (\pi/180.0)] \cdot \cos[\theta \cdot (\pi/180.0)] + \Delta y, \\
z &= \gamma \sin[\phi \cdot (\pi/180.0)] + \Delta z
\end{aligned}
\qquad (5.1)
$$

where, Δx, Δy, and Δz are the x, y, and z displacements of the location of a particular radar station from the mid-point of the four CASA radars. After the coordinate conversion, we preprocess raw radar measurements.

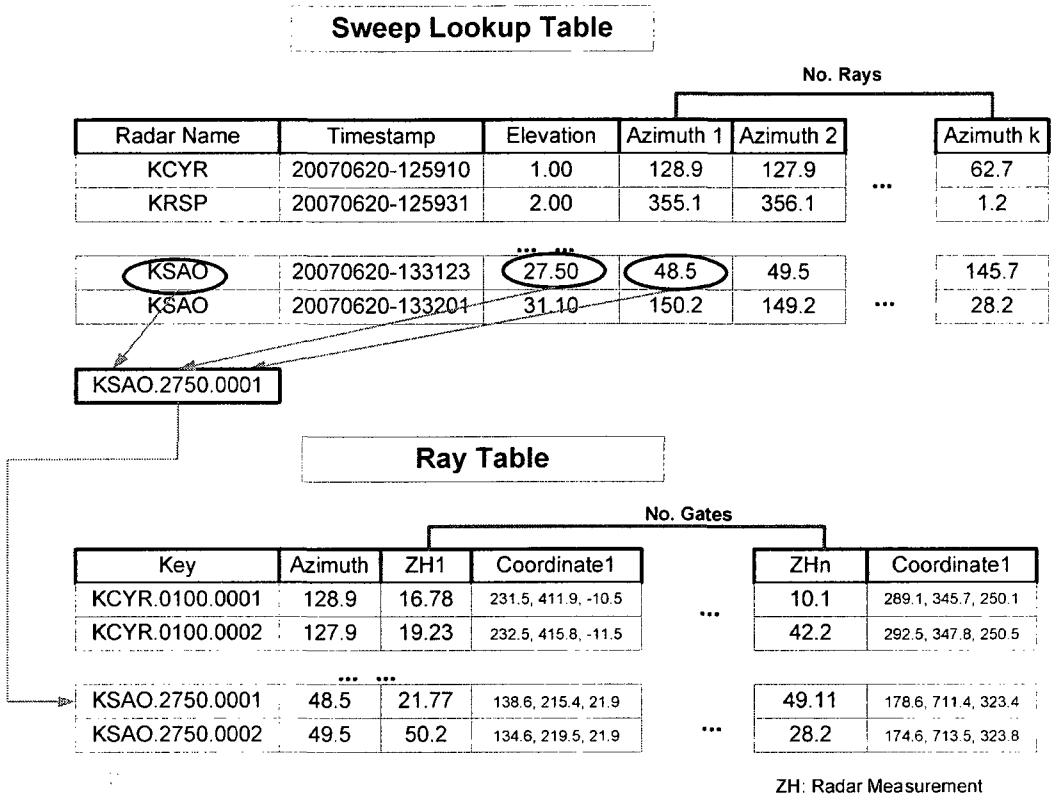


Figure 5.2 Sensor database index structure

The preprocessing can be as simple as cleanup (thresholding) of the data or as complex as the use of computing intensive digital signal processing approaches, depending on applications.

The Cartesian coordinate and the preprocessed results are loaded into the sensor database. The multi-sensor fusion applications algorithm processes radar data on a ray-by-ray basis, thus it is desirable to allow the applications to select a set of individual rays to suit their needs. In order to provide rapid random lookups and efficient access of individual rays in the database we develop an index structure. Fig 5.2 illustrates the index structure of the sensor database. The index structure consists of a sweep lookup table and a ray table. The sweep lookup table is created in the main memory. The sweep lookup table is used by the common volume finding module to find a particular ray with a specific elevation angle and an azimuth angle. Each of the sweep lookup table entity holds a radar sweep description, such as source radar name, timestamp denoting the time when the radar sweep is captured, an elevation angles and azimuth angles. In turn, we create

a ray table that holds the preprocessed radar measurements, the coordinates of data points, and the number of gates, etc. in the database. As seen in Fig 5.2, there is an additional column called key. We use this column as a primary key, i.e. a unique identifier of each row in the ray table. A primary key is generated by combining the source radar name, the elevation angle, and the ray index number in a radar sweep. In Fig.5.2 the key value 'KSAO.2750.0001' can be interpreted such that the source radar is KSAO, the elevation angle at which the sweep is captured is 27.50 degree, and '0001' means is the 1st ray of the sweep. Since the data management module keeps only the recent sweep that is collected at a particular elevation angle for individual radars, the primary key can uniquely identify individual rays in the database. The sweep lookup table contains a relatively small amount of information, thus it is possible to locate the table in the main memory of the system without affecting the performance of the other applications within the same system. On the contrary, the ray table contains actual radar measurements and other radar operational parameters; therefore it requires maintaining a large amount of data. We store the ray table in a relational database system. Without the index structure, the common volume finding module has to start with the first record and then read through the whole database to find the relevant rays. In Section 5.3, we will discuss how the common volume finding module accesses the database in a time-effective manner using the index structure.

5.3 Common volume finding module

We now present the common volume finding module that is used for finding a set of correlated data items for multi-sensor data fusion. Finding a common volume considering the spatial and temporal correlation in real-time is a very challenging task as each of the radars generates tens of megabyte-scale data per minute.

```
find_common_volume
(
 ITEM_List          itemList,  /* list of points of interest given by fusion application */
  float             pdist,     /* acceptable physical distance  given by fusion application */
  float             timegap    /* acceptable generation time gap given by fusion application */
 )
{
  int       i;                          /* index variable */
  int       j;                          /* index variable */
  int       correlated_item_count;      /* correlated item count */

  for each item in itemList[] i  {
        correlated_item_count = 0;

        for each radar in operational_radar_list[] j {

        /* find a sweep table entity */
        Record el = find_best_match_elevation(operational_radar_list[j], itemList[i]);

        /* find a ray  index value from the entity */
         int     inx = find_best_match_azimuth(operational_radar_list[j], itemList[i], el);

        /* generate database key (radar name, elevation angle, ray index ) */
        bd_key = key_gen(operational_radar_list[j].name,el.elevation, inx);

        /* read database */
        Ray ray = read_database(db_key);

        if ( ABS(ray.gentime-itemList[i].gentime) < timegap)) { /* check data generation time */
                int       k;              /* index variable */
                float     min = -1.0;     /* initialize with -1 */

                for each gate in ray.gate[] k {
                        /*  DISTANCE(p1, p2):Euclidian distance between p1 and p2 */
                        float diff = DISTANCE(ray.gate[k].coordinate, itemList[i].coordinate);

                        if (min == -1.0) {
                                min = diff;
                                best_matched = k;
                        }
                        else {
                                if (min  >  diff) {
                                        min = diff
                                        best_matched = k;
                                }
                        }
                }

                /* acceptable physical distance ? */
                 if (min < pdist) {
                        /* add the item to the correlated item list */
                        itemList[i].correlatedItem[correlated_item_count++] = ray;
                }
               itemList[i].correlatedItemCount = correlated_item_count++;
               correlated_item_count = 0;
        }
  }

  return;
}
```

Figure 5.3 Common volume finding algorithm

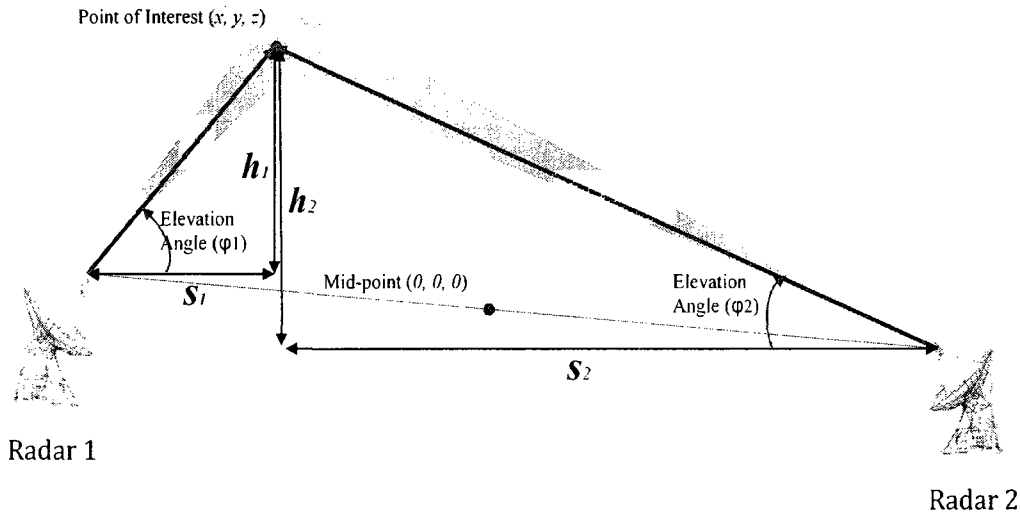


Figure 5.4 Schematics of calculation of elevation angle

In order to address the problem, we provide a query processing mechanism that uses the database index structure and trigonometric equations. In designing the common volume finding module, we assume that each of the multi-sensor data fusion applications selects the points of interest in the atmosphere and specifies the points using the Cartesian coordinates. Furthermore, an application has its own data selection criterion in terms of the maximum tolerance in temporal and spatial differences among the data to be integrated. After choosing a set of points of interest, a data fusion application passes a request to the common volume finding module with the application-specific data selection criterion to select a set of common volumes. Fig 5.3 is the pseudo-code for the common volume finding algorithm. As illustrated in the figure, the first step in finding common volume is to estimate an elevation angle that could be relevant to a point of interest. Fig. 5.4 shows the schematic of calculation of the elevation angle. In the figure, the coordinates of a CASA radar station can be obtained by converting the latitudes, longitudes and altitudes of the radars to the three-dimensional Cartesian coordinates.

```
Record find_best_match_elevation
(
   RADAR radar,  /* radar information (location, coordinates ...) */
   ITEM item         /* a point of interest specified by fusion app.    */
)
{
   int         j;                        /* index variable */
   int         best_matched;        /* best matched elevation angle index */
   float       RA2DE = 180/PI;     /* radian to degree conversion */

  float       dx = item.x – radar.x;
  float       dy = item.y – radar.y;

  float       s = sqrt(dx*dx + dy*dy);
  float       el = RA2DE * atan((item.z – radar.z)/s); /* elevation angle estimation
*/
  float       min = -1.0;                    /* initialize with -1 */

  for each record in radar.sweep_lookup_table[] j {
    /* ABS(a) : absolute value of a*/
    float diff = ABS(radar_sweep_lookup_table[j].elevation – el);

          if (min == -1.0) {
                    min = diff;
                    best_matched = j;
          }
          else {
                    if (min  >  diff) {
                              min = diff
                              best_matched = j;
                    }
          }
   }

    return radar.sweep_lookup_table[best_matched];
}
```

Figure 5.5 Best match elevation angle finding algorithm

By applying a simple Euclidean distance calculation, we can find ***s***, the distance between a radar station and a point of interest specified by the application along the x-y plane of the coordinate system. Similarly using the coordinate of the point of interest and radar's coordinate, we can easily find ***h***, the vertical distance from the point of interest to the radar. Thus, the elevation angle Φ at which the data point captured by the radar can be represented using the following equation:

$$\phi = \arctan(|\, h / s \,|) \qquad (5.2)$$

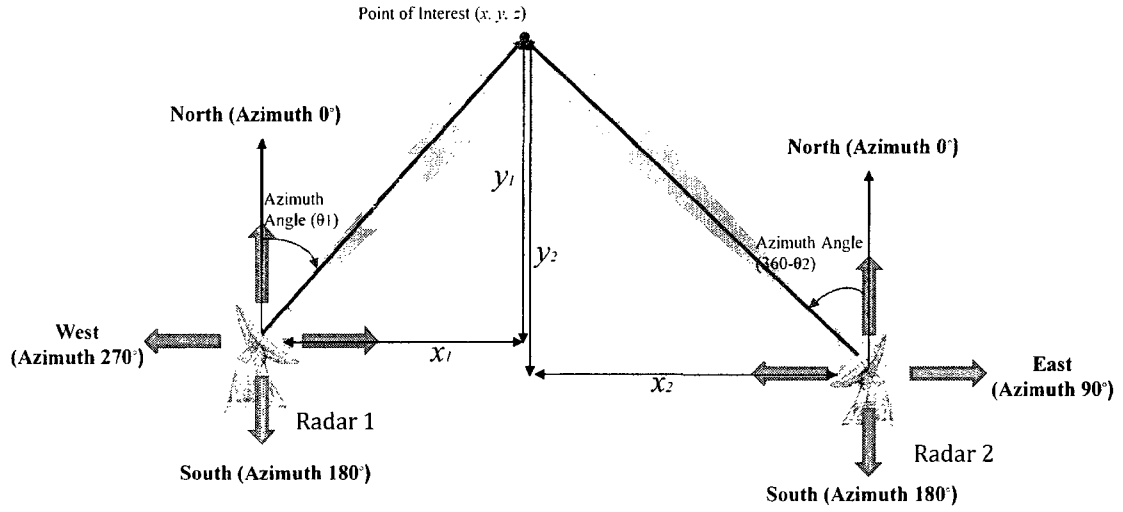


Figure 5.6 Schematics of calculation of Azimuth angle

After obtaining the estimated elevation angle, the common volume finding module searches the sweep lookup table to find an entity that has the best match elevation angle with the estimated angle using (5.2). The pseudo-code in Fig. 5.5 describes this procedure.

As we mentioned in Section 5.1, the sweep lookup table is constructed in the main memory and each radar has less than 20 sweeps per heart-beat interval in the CASA network. Thus, the time required to find the best match elevation angle in the sweep lookup table is minuscule. Similarly, we derive another trigonometric relation to find an azimuth angle θ. Fig. 5.6 illustrates the schematic of calculation of the azimuth angle. The pseudo-codes in Fig.5.7 describe the procedure for finding the best match azimuth angle. The CASA radars have hundreds of different azimuth angles in a sweep at the most, so finding the best match azimuth angles in the sweep lookup table entity that is found by the procedure in Fig 5.7 takes just few micro-seconds. By combining source radar's name and the best match elevation and azimuth angles, we can generate a database key that identifies a particular ray in the sensor database. The common volume finding module reads ray data from the database using the database key.

```
int find_best_match_azimuth (RADAR radar, ITEM item, Record record )
{
   int        j;                    /* index variable */
   int        best_matched;     /* best matched azimuth angle index */
   float      RA2DE = 180/PI;   /* radian to degree conversion */

  float      dx = item.x – radar.x;
  float      dy = item.y – radar.y;

  float      az = RA2DE * atan(dy/dx); /* azimuth angle estimation */
   float      min = -1.0;            /* initialize with -1 */

   if (dx > 0 && dy > 0) az = 90 - az;
   else if (dx < 0 && dy > 0 ) {
         az = -1 * az;
         az = az + 270;
   }
   else if (dx < 0 && dy < 0) {
         az = 270 - az;
   }
   else {
         az = -1 * az;
         az = az + 90;
   }

  for each value in record.azimuth[] j {
         /* ABS(a) : absolute value of a*/
         float diff = ABS(record.azimuth[j] – az);

         if (min == -1.0) {
                  min = diff;
                  best_matched = j;
         }
         else {
                  if (min  >  diff) {
                           min = diff;
                           best_matched = j;
                  }

         }
    }

    return best_matched;
}
```

Figure 5.7 Best match Azimuth angle finding algorithm

As the final step to finding common volume we check whether the selected ray is within acceptable temporal and spatial distances based on the criterion given by the application.

5.4 Algorithm processing module

In this section, we describe a multi-radar data fusion algorithm, the network-based reflectivity retrieval algorithm. Because the algorithm processes a sweep on a ray-by-ray basis, it is possible

to run multiple identical threads simultaneously to process multiple rays at the same time. We implement the algorithm processing module using multiple identical threads. In addition, we also describe the details of the module and the logic behind our design choices.

5.4.1 Network-Based Reflectivity Retrieval (NBRR)

The network-based reflectivity retrieval (NBRR) is established on simultaneous observations of the same event by multiple radars in different locations. This type of radar collaboration is illustrated in Fig.5.8. In the absence of attenuation, the reflectivity in a common volume for each radar should be the same. However, the reflectivity observations in a common volume are different because of the difference of the integrated attenuation along the paths from each radar. Specific attenuation distribution can be solved by the integral equation for reflectivity with total path attenuation constraint, in a manner similar to that used with a differential phase constraint [Lim07]. In the NBRR algorithm, a radar sweep acquired or received from the reference radars (Radar B and C in Fig. 5.8) are used as supplementary information for correcting sensing errors in a sweep collected by the base radar (Radar A in Fig 5.8). The NBRR algorithm iterates the following steps over each ray in a sweep in order to correct sensing errors in the sweep:

1) Sample a number of gates from the current ray. A ray consists of hundreds of gates. The NBRR algorithm needs to select 10~20 gates which are evenly sampled from a ray in order to operate properly.
2) Find a common volume for each sampled gate. The sampled gates are given to the common volume finding module as the points of interest. The common volume finding module chooses a set of rays from the reference radars for each of the selected gates considering the temporal and spatial distance.
3) Correct sensing errors in the current ray using reference radars' rays selected by the common volume finding module.

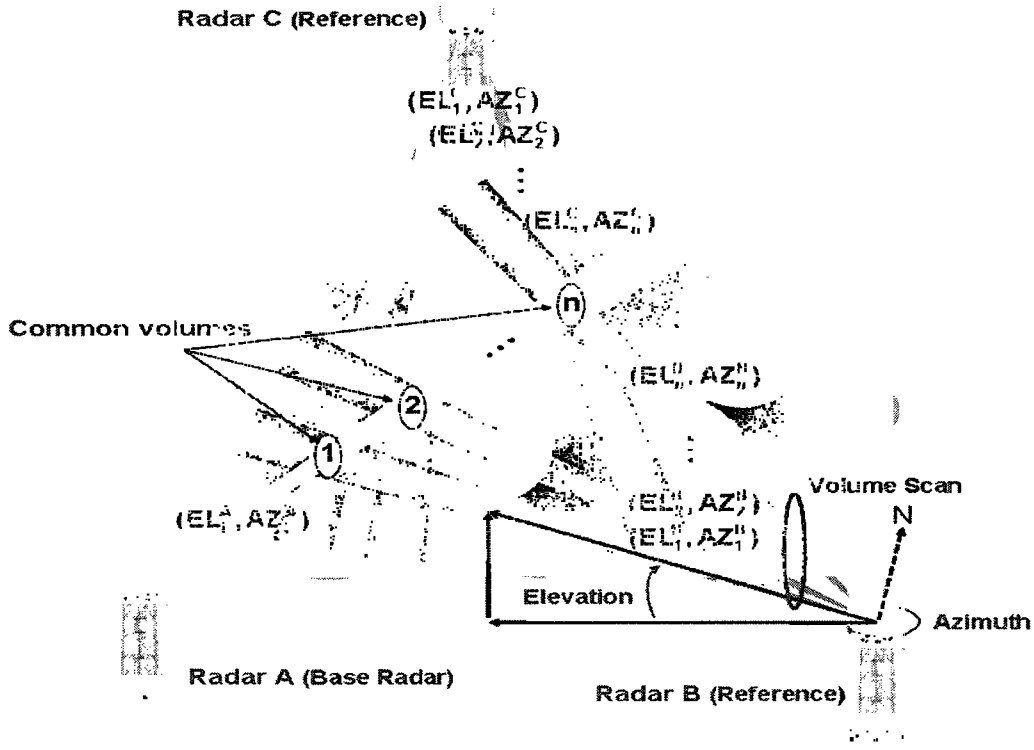


Figure 5.8 Collaborative radar operation [Lim08]

In each iteration of the algorithm, there is no dependency among the rays from the same sweep.

5.4.2 Parallelizing Fusion Algorithm Processing Module

Radar algorithms can benefit from multi-core or multiprocessor systems. To take advantage of multi-core or multiprocessor architecture, there are two obvious requirements: first, the operating system must support the multi-core architecture by providing a scheduling mechanism to allocate processes and threads in accordance with CPU status, the priority of threads currently being processed, and other factors; and second, the algorithm should be written to utilize multiple cores or CPU. Many of the currently available operating systems, such as GNU/Linux, Windows XP, and Mac OSX support kernel-level threading where threads are managed by the kernel, and they

have access to all kernel resources including multi-core or multiple processors. Thus, several threads belonging to the same process can be concurrently active, each with its own processor or core, so that true parallel activity occurs [Da00]. In order to instruct the operating system to execute a number of independent tasks at the same time on the multiple cores or processors, explicit threading using the standard POSIX thread library is one of the best options. The implementation of the algorithm using multithreading requires a careful consideration of the complexity of the approach and a thorough analysis of the resulting performance. As we discussed in Section 5.4.1, the examination of the NBRR algorithm shows that each ray in a sweep can be processed independently and individually. Because a number of independent rays can be processed simultaneously through the same operation, the NBRR algorithm is suited to data parallelism. According to Amdahl's Law [Am67], the speedup of parallelized implementation of an algorithm is decided by the fraction of the program that can be performed concurrently. This means the parallelized implementation can result in significant speedups when the fraction of parallelization is large enough; otherwise it yields a small return with extra programming complexity and communication overhead. Therefore, it is important to decide whether or not to develop a parallel version of a program considering the expected speedup of parallelized implementations of the algorithm. We measure the fraction of the execution time taken by the algorithm processing module. Fig. 5.9 shows the percentage that each module of the software contributes to the total execution runtime of data fusion processing. These execution runtimes were measured in the CASA IP-1 testbed while a severe storm was moving through the testbed's coverage area. As seen in the figure, approximately 70% of the execution runtime is spent on data fusion algorithm processing. Meanwhile the other two parts, finding common volume and updating database take up around 30 % of the total processing time. The algorithm processing module consumes the greatest part of total execution runtime. In addition each ray in a sweep can be processed independently by a thread. Therefore the multi-radar data fusion algorithm is favorable to thread-level parallelism.

Multithreading approaches often require synchronization that occurs whenever threads read or write to a shared data structure. In order to have the threads perform the most work with the least amount of the synchronization overhead, each of the algorithm processing threads has its own message queue to receive selected common volumes from the common volume finding module. Fig 5.10 illustrates the simultaneous operation of the multiple threads. In the figure, the common volume finding module continues to select common volumes on a ray-by-ray basis and forwards them to the algorithm processing threads to be processed. In order to distribute the processing overhead of the algorithm over the multiple threads, the common volume finding modules use a load distribution strategy based on the queue sizes of the algorithm processing threads. When the common volume finding module completes selecting common volumes for a ray, it inserts the common volumes to the message queue of algorithm-processing thread that has the minimum number of messages in its queue.

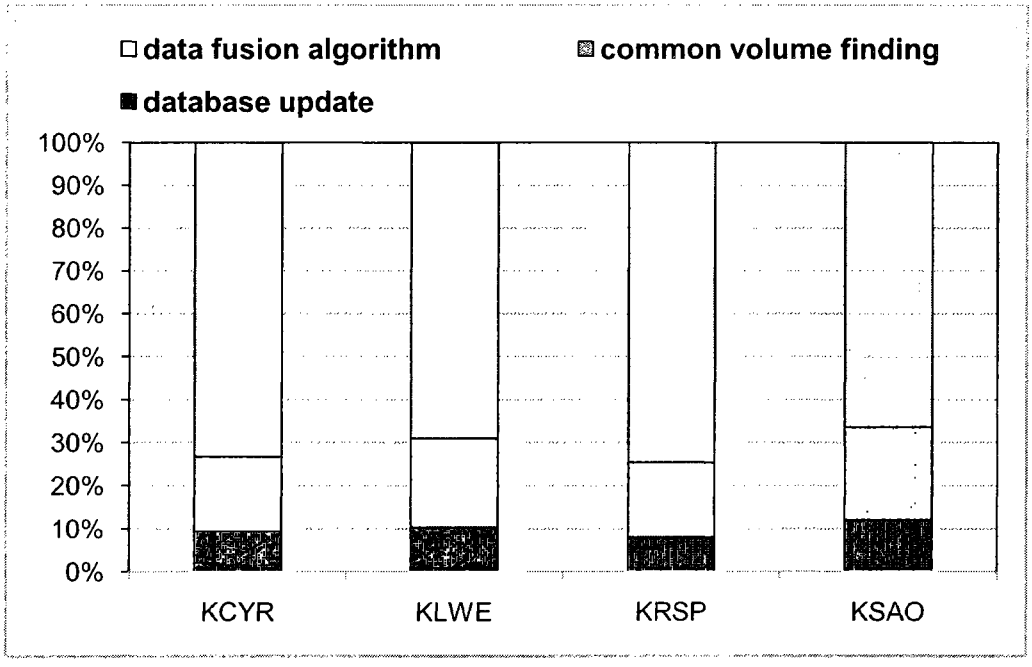


Figure 5.9 Percentage that each module contributes to the total execution runtime

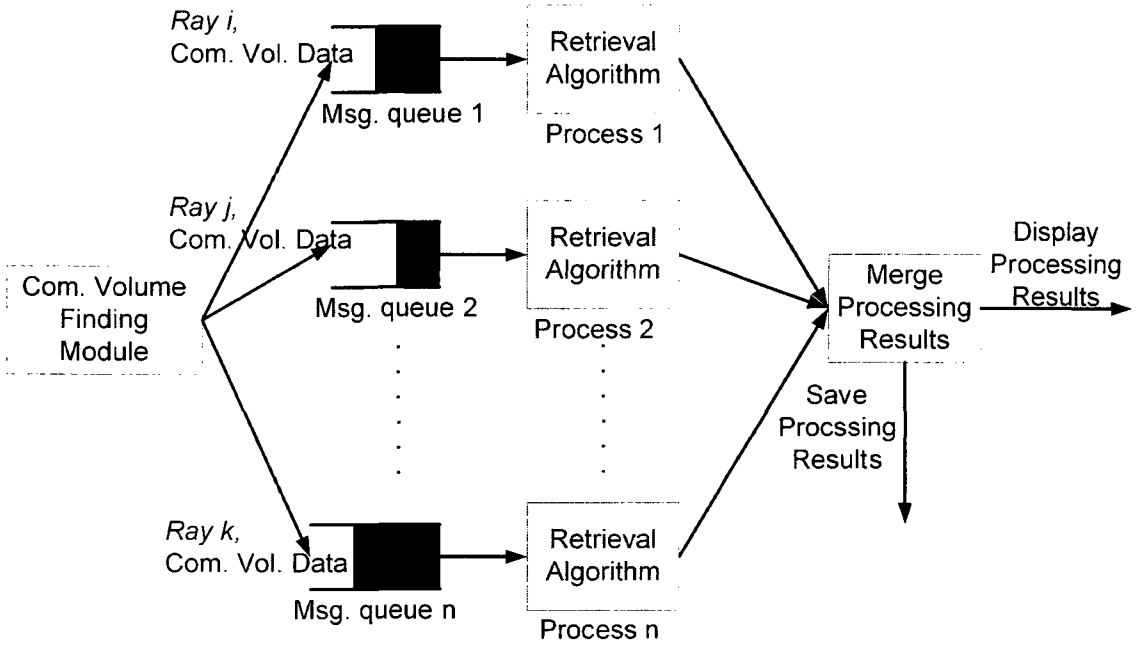


Figure 5.10 Simultaneous operation of multiple algorithm processing threads

Another important factor we need to consider is the number of threads to use for the algorithm processing module. The optimal number of threads to use depends upon the size and characteristics of the meteorological objects as well as the number of processors or cores we have. In order to determine the number of threads to use for the algorithm processing module we measure the execution runtime of the program with 1, 2, 4, 8 and 12 radar algorithm processing threads using data collected while a severe storm is moving through the testbed's coverage area. In small storm scenarios the performance of the software is not dependent on the number of threads in use. The result of running the program with different numbers of threads is shown in Fig. 5.11. The figure illustrates per-sweep execution runtimes including database update, common volume finding and algorithm processing. The machine used in the experiments is equipped with two quad-core Intel Xeon processors (two sockets in a node and four cores in a socket), each operating at 2.83 GHz. The total memory in the machine was 8GB.

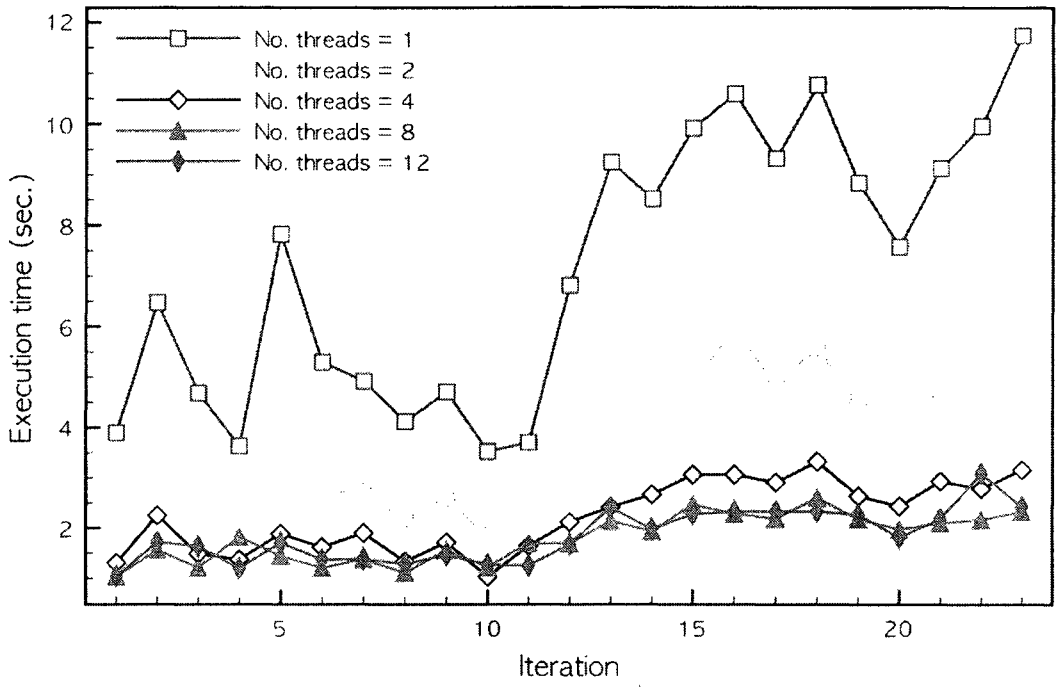
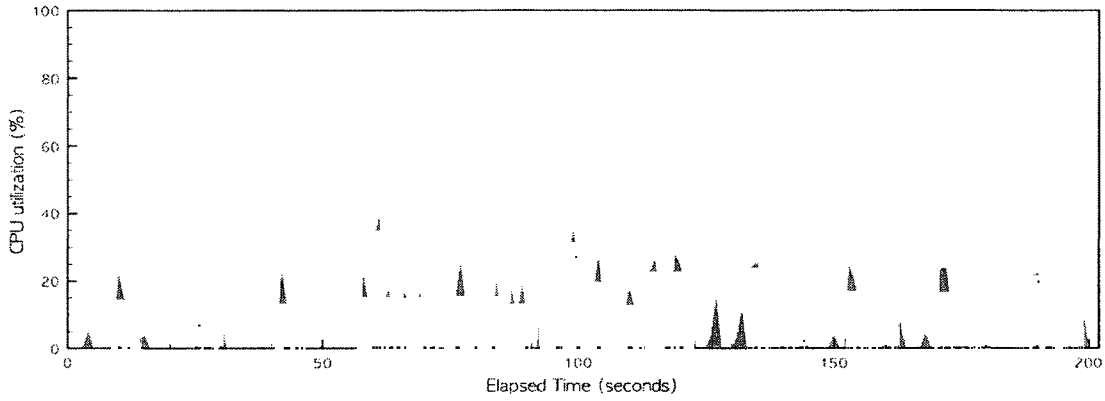
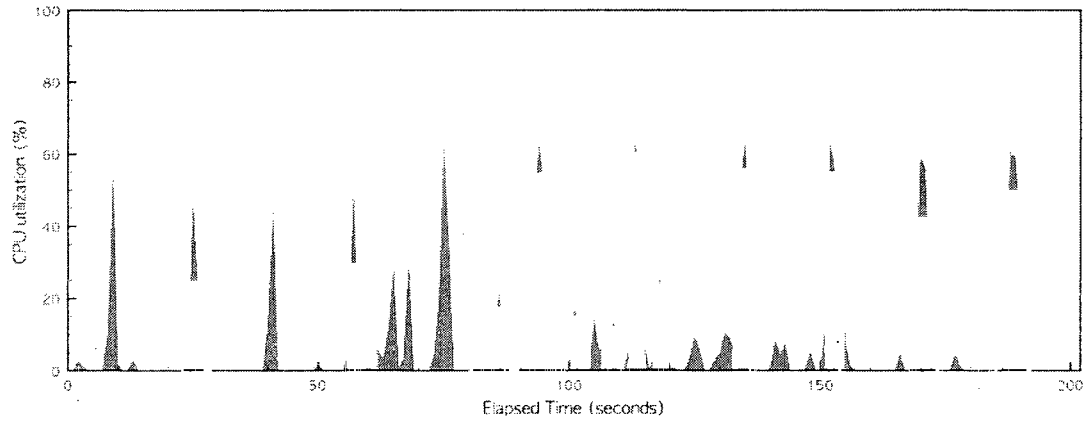


Figure 5.11 Execution runtime with varying the number of threads

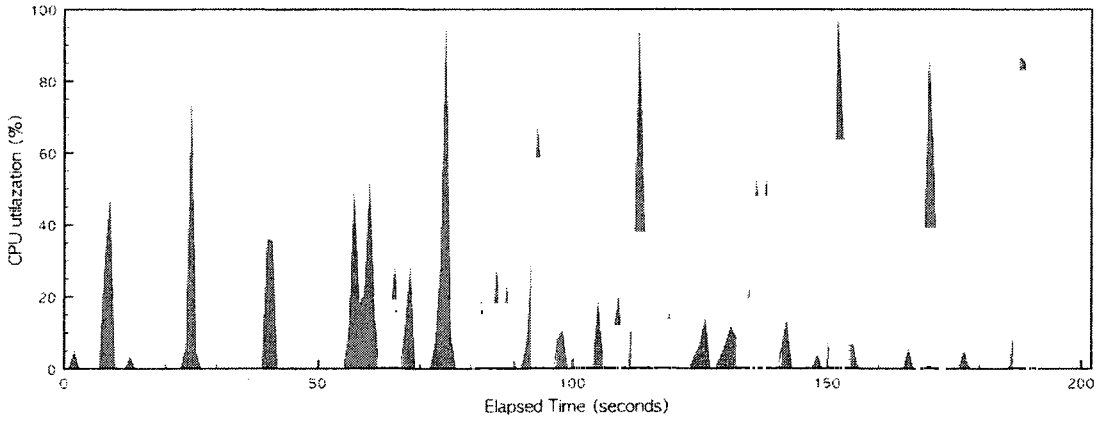
The operating system running on the machine was GNU/Linux with 2.6.18 kernel. The number of algorithm processing threads significantly impacts the execution runtime of the program in the storm case. Per-sweep execution runtime decreases as the number of algorithm processing threads increases. The results demonstrate that the eight-thread and twelve-thread cases show improved performance. As illustrated in the figure, the difference between eight-thread and twelve-thread cases is not significant, so we fix the number of algorithm processing threads to eight. To better illustrate the effect of the number of algorithm processing threads, we measured the CPU utilization using the mpstat tool [Sys] that reports CPU utilization calculated as averages among all processors or cores. Fig 5.12 plots the CPU utilization varying in the number of threads. While running the program, a CPU utilization value is reported every 1 second.



(a)



(b)



(c)

Figure 5.12 CPU utilization for one (a), four(b), and eight(c) algorithm processing threads

The one-thread case shows lower CPU utilization with long CPU occupation time that results in the large execution runtime. As we increase the number of threads, the CPU utilization increases and CPU resource is effectively aggregated for the multi-radar data fusion as illustrated in Fig. 5.12(b) and Fig. 5.12(c).

5.5 Performance Evaluation

In this section, we present empirical measurements of the execution runtimes of the NBRR program. We use CASA radar data as inputs to the program. The data were collected from CASA IP-1 network during the periods of time when storms were moving through the network coverage area. The data generated from all four IP1 nodes were processed by a single machine simultaneously. The CASA radar scans are controlled by the end-user requirements and the meteorological features present in the radar coverage area. The types of scans that are implemented in CASA IP-1 testbed are mainly sector PPI scans at multiple elevations and a surveillance scan at an elevation angle of 2 degree. Table 5.1 describes the radar data that were used for the performance evaluation. The data in cases 1 and 2 were collected from all four IP1 sites (KCYR, KSAO, KRSP and KLWE). Each volume scan of the cases consists of a full 360 degrees scan at 2 degree elevation angle and sector scans collected at 1.00, 3.86, 6.70, 9.50, 12.26, 14.96, 17.60, 20.47, 23.72, 27.34 and 31.31 degrees elevation angles. In these cases volumes were taken every 170 seconds.

The data in cases 3, 4 and 5 were collected by all four radars. In these cases, the radar scans were performed with 60 (20+40) seconds heartbeat interval where 20 seconds were used for full 360 degrees scan at 2 degree elevation angle while the rest 40 seconds were used for sector scans at 1, 3, 5, 7, 9, 11, and 14 degrees elevation angles.

Fig. 5.13 shows the histogram of per-sweep execution runtimes for Case 1, Case 3 and Case 5 data sets. As seen in the figure, 69% of the sweeps in Case 1 and 62% of the sweeps in Case 5 were executed in sub-seconds. More than 98% of the sweeps in both cases were processed in 5 seconds. The software took a longer time to process Case 3 data set. In Case 3, only 40% of sweeps were processed in sub-seconds, and the execution runtimes for approximately 5% of the sweeps were larger than 5 seconds.

The variations in the execution runtime result from the different sizes of the meteorological objects in the radars' field in the cases. In Case 1 and Case 5, the events were isolated storms with relatively small coverage area, whereas in Case 3, several severe storms covered most of the IP1 network area, and the storms continued to intensify. In addition, each CASA radar had a 30km maximum range in Case 1 and 2, but the maximum range increased to 45km in Case 3, 4, and 5. Because the radars collected larger amounts of data with the increasing maximum range, the software took more time for preprocessing raw radar measurements, updating database, and finding common volume due to increasing radar overlapping regions.

In order to prevent the systems from overflowing, it is crucial to maintain the average execution runtimes for the software below the average radar sweep generation interval. Table 5.2 illustrates the average sweep generation intervals and the average per-sweep execution runtimes for the test cases. Fig. 5.14 shows the data corresponding to the results shown in Table 5.2. Our measurements demonstrate that approximately one and two seconds per-sweep processing times are needed on average for isolated storm cases (Case 1, Case 2 and Case 5) and clusters of storms cases (Case 3 and Case 4) respectively. When the radars perform the surveillance scans at 2 degree elevation angle, the amount of overlapping area and the size of data collected by the radars are maximized (Recall that other higher elevation scans are sector scans). Thus, the sweeps captured at 2 degree elevation angle had larger execution runtimes in general. The average runtimes for the two-degree surveillance scans for Case 3 and 4 are larger than 3sec.

Test Case	Time of Event	Event Description	Maximum Range	No. of Sweeps
1	06/10/2007 20:00-24:00 UTC	An isolated convective storm	30km	3773
2	06/16/2007 19:00-23:00 UTC	An isolated storm	30km	3870
3	05/07/2008 08:30-12:45 UTC	Several severe storms moved through the network	45km	4416
4	05/27/2008 06:30-11:30 UTC	Cluster of storms entered the network	45km	6339
5	06/01/2008 03:30-07:30 UTC	An isolated super-cell	45km	4273

Table 5.1 Test data sets

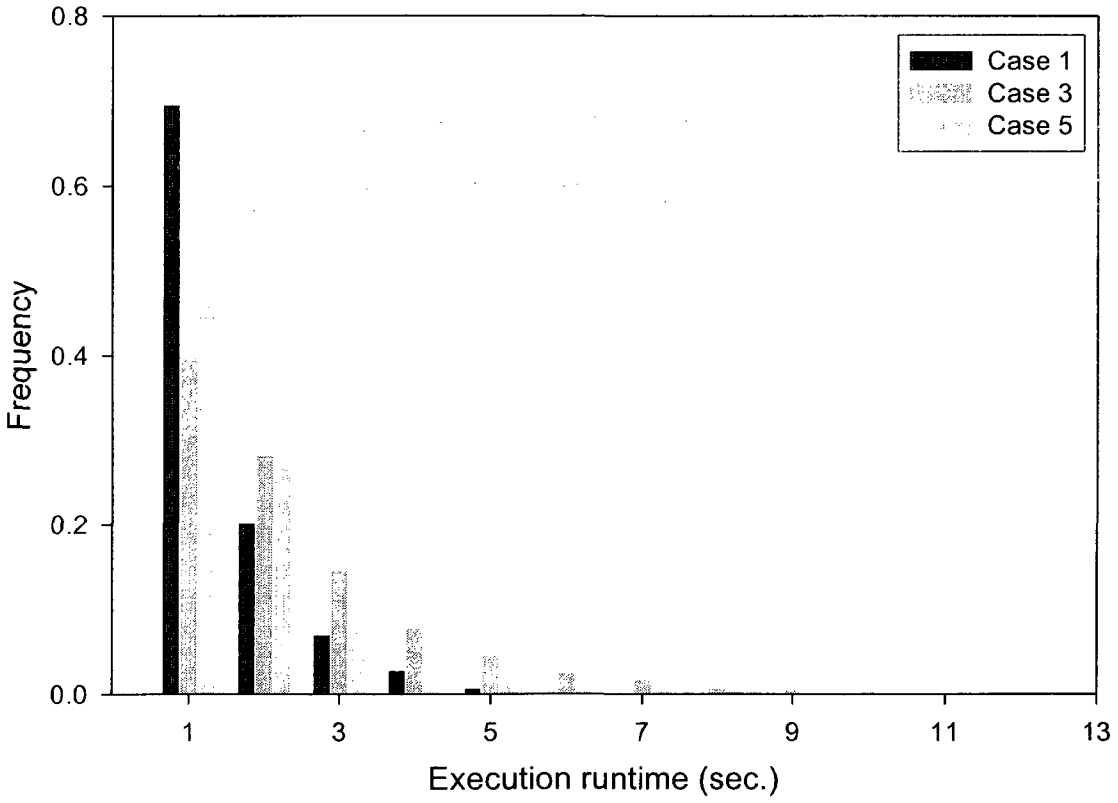


Figure 5.13 Per-sweep execution runtime histogram

TEST CASE	AVERAGE SWEEP GENERATION INTERVAL	AVERAGE EXECUTION TIME PER SWEEP	EXECUTION TIME STANDARD DEVIATION	MAX EXECUTION TIME
1	3.81SEC.	0.95SEC.	0.81	5.89SEC.
2	3.71SEC.	0.74SEC.	0.59	5.75SEC.
3	3.46SEC.	1.83SEC.	1.59	11.33SEC.
4	2.83SEC.	2.01SEC.	1.77	11.06SEC.
5	2.87SEC.	1.09SEC.	0.95	9.18SEC

Table 5.2 Execution runtimes for the test cases

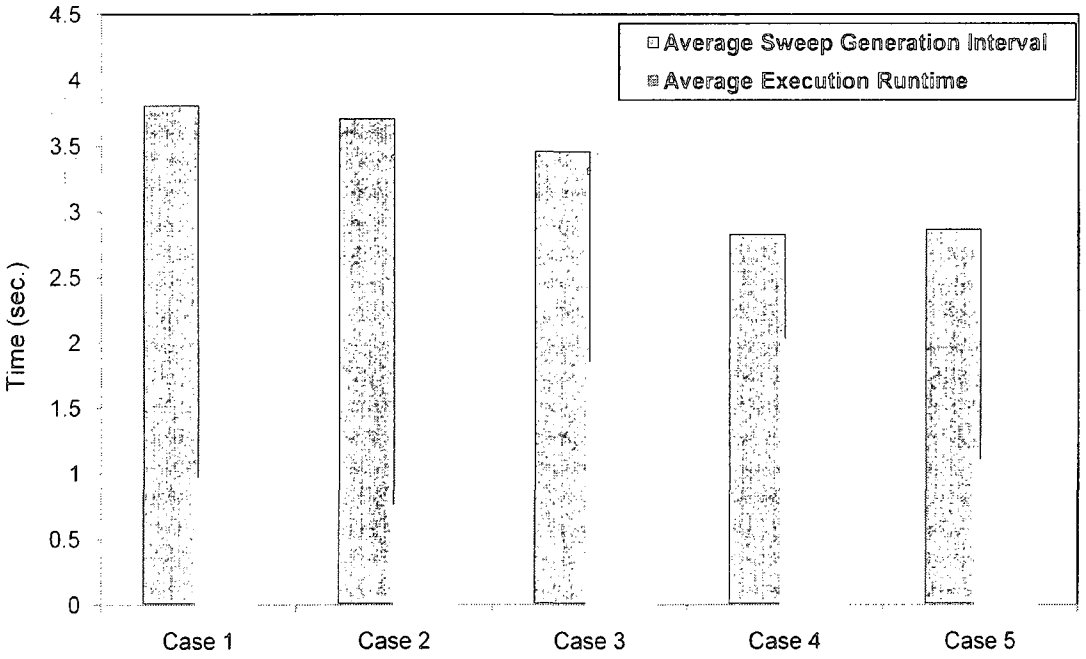


Figure 5.14 Comparison of average sweep generation time and average execution runtime

However, the software took a significantly less time to execute sector scanned sweeps (e.g., average 0.68sec. for 14 degree scan in Case 3). Thus, the average execution runtimes can be maintained below the average sweep generation intervals for all the test cases.

5.6 Summary

We have presented the design and implementation of an architectural framework for timely and accurate processing of radar data fusion algorithms in CASA IP1. The framework provided a data managing and searching mechanism and a common volume finding mechanism. In addition, it accommodated simultaneous and collaborative operation of multiple threads to meet the execution time requirements of the CASA multi-radar data fusion application. Using the framework we implemented a CASA multi-radar data fusion algorithm, the network-based reflectivity retrieval algorithm in CASA-IP1. Through the execution runtime measurements of NBRR over several storm cases, we have showed that the framework could meet the real-time requirements of the application. On-going work includes providing an adaptive data synchronization mechanism considering the difference in network delays from different radars. In the current data synchronization strategy, the common volume finding model picks a set of data items arrived in the past time in the sensor database. Because the common volume finding module considers application-specific data synchronization criterion, the data generation time difference among the selected data item is satisfactory. In the CASA network, each of the radars may be connected to different networks that have different available bandwidth and other network properties, so network latencies from the individual radars to MC&C are different. Therefore, in order to obtain more closely time-matched data set, it is desirable to develop an adaptive data selection scheme with the consideration of the difference in network delays.

Chapter 6

A PEER-TO-PEER COLLABORATION FRAMEWORK FOR MULTI-SENSOR DATA FUSION

Recent advances in technology have made it possible to develop distributed collaborative adaptive sensing (DCAS) systems, which are revolutionizing how we observe, evaluate, understand and predict hazardous weather events. These systems use a large number of distributed, powerful, high-data rate sensors, such as radars, to improve spatial and temporal resolution throughout the sensing area. In addition, such systems operate the sensors collaboratively, and adapt them to changing conditions in a manner that meets competing end-user needs [Mc05][Ku06]. Unlike mote-based resource limited wireless sensor networks, which are designed to carry out one or few tasks while minimizing energy expenditure, DCAS systems are intended to serve a variety of applications and users with different requirements [Li07]. Due to the inherently high cost of radar sensors, the massive amount of data generated by such sensors and the computing intensive processing required by applications, these sensor/processing nodes are usually equipped with significant communication and computation resources.

For automated identification of the hazardous weather features, the conventional stand-alone radar systems relied on a suite of meteorological feature detection algorithms using data from a single-radar. Achieving improved accuracies and more specific inferences with emerging multiple radar DCAS based solutions require combining data from the multiple radar sensors using multi-radar data fusion algorithms [Lim07]. Client-server is a commonly used framework

for realizing multi-radar data fusion, especially when the data rates involved are high. In a client-server framework, a powerful server acquires data from sensors to perform the algorithms. Although widely used, such a framework is not scalable, and is not appropriate as the demand on the server would increase in proportion to the total number of sensors, quickly overrunning server's limited capacity; it may also cause network congestion close to the server. Peer-to-peer (P2P) framework is thus a very attractive alternative paradigm to the client-server architecture for DCAS systems. In contrast to client-server frameworks, each node in peer-to-peer networks can provide bandwidth, storage and computation. For example, in file sharing P2P networks, such as BitTorrent (http://www.bittorrent.com/), Napster (http://www.napster.com) and Gnutella (http://gnutella.wego.com), each peer supplies disk space to store the community's collection of data and bandwidth to move the data to other peers, and obtains others' resources in return. When communication cost becomes too high (e.g., in case of large multimedia content transfers), P2P networks can avoid the bandwidth limitation problem by spreading the cost over several peers. Likewise, resource-intensive sensor applications can benefit from P2P computing. In P2P frameworks, the applications can aggregate and utilize unused resources from peers over the network to achieve better performance on processing sensor data. Furthermore, the P2P mechanism also reduces the risk of a single-point-of-failure, such as possible with the client-server architecture; such a failure can be catastrophic in CASA like systems that have to perform critical functions under hazardous weather conditions, which can potentially interfere with some of the system infrastructure.

To use a P2P framework as a feasible solution for multi-radar data fusion, we must take the mission-critical nature of the sensor applications into account. With multi-radar data fusion, the peers need to exchange data to be integrated; thus the sensor measurements or data are dynamically replicated among the peers participating in the processing. Since geographically distributed sensors generate a substantial volume of data, the communication cost cannot be ignored. In addition, the large volume of generated data must be processed within stringent time

constraints, so vast amount of computation resources have to be deployed on the fly. Furthermore, in these high-end sensor networks, the peers joining the multi-radar fusion algorithm can have a wide range of network access speeds, available bandwidths, and variability in system load. Therefore, the main problem is to coordinate a group of peers, each with its performance limitations and user demands. That is, when several sensor nodes work together on multi-radar data fusion processing, they need to decide which node would be responsible for providing particular contents with acceptable performance for the multi- sensor data fusion processing. Thus, the decision must involve not only locating peers who have the desired contents, but also selecting a peer that can provide the contents within the real-time constraints. Another important issue to be considered is how to select a set of data from multiple sensors when the sampling time and intervals of the sensors are not synchronized. In a radar sensor network where each of the sensors can be assigned a certain sensing task with different mode and sensing interval, the synchronization of data generation time is a difficult task. In the presence of imperfectly synchronized data generation, we need to provide a mechanism that notifies the peers participating in multi-radar data fusion processing when new data are available for processing. Furthermore, a scheme is needed that selects the appropriate set of data considering the time gap between data and timeliness requirement of processing.

In this chapter, we present a P2P collaboration framework for CASA multi-sensor data fusion algorithms. We also provide a data selection mechanism that addresses the problem associated with the data synchronization. Furthermore, we propose Best Peer Selection (BPS), a dynamic peer selection algorithm that allows the multi-sensor data fusion algorithm to locate the desired contents in a scalable, efficient, and distributed manner, and to obtain the desired contents minimizing time. In our radar sensor network, each of the peers maintains a list of peers which are participating in the same multi-sensor data fusion process. The required contents are located by querying peers on the list. Upon receiving the query, the peers estimate the time required for providing the desired contents considering their computation and communication overhead, and

respond to the query-initiator node with the estimated time. Based on the response messages, BPS can find a subset of the peers that can collectively provide the given set of contents within the time constraint. We demonstrate the feasibility and efficiency of BSP using simulations. The experimental results show that BPS can deliver a significant performance improvement, especially when the peers and the network have extremely high variability in resource availability.

The rest of this chapter is organized as follows. Section 6.1 we introduce the target system and outline its salient features that are essential for describing the data fusion strategy. The node architecture is presented in Section 6.2. Section 6.3 presents the data synchronization mechanism. Section 6.4 details BPS, and Section 6.5 presents the experimental results. Section 6.6 summarizes this chapter.

6.1 System Model

Fig. 6.1 illustrates collaborative and adaptive radar operation and multi-sensor data fusion. The radars collaborate on sensing and detecting weather events by focusing their scans on overlapping region in the atmosphere. It is often beneficial to have simultaneous observations of the same event by multiple radars in different locations [Ku06][Lim07] as the observations from multiple radars offer different, but complementary views of the same weather features. In this multi-radar collaborative environment, multi-radar data fusion applications are increasingly common and essential constituents of the systems. In Fig. 6.1, the required computations for the algorithm are distributed to multiple participant nodes to exploit parallel/distributed processing. This section identifies challenges in multi-radar data fusion using a P2P collaboration framework in the CASA network.

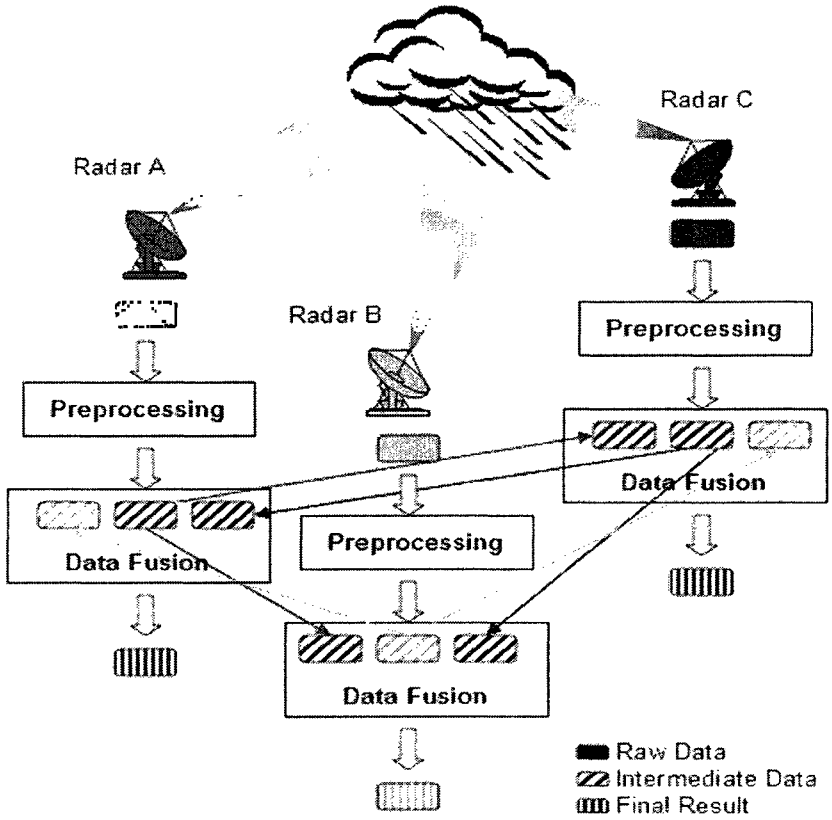


Figure 6.1 Multi-radar data fusion

6.1.1 Data Generation Timing

In the CASA testbed network, some radar scan tasks are triggered only when a meteorological feature is detected in the coverage area, while other tasks, such as surveillance scans at lower elevations, are triggered periodically. In addition, the various users are able to express their sensing preferences in terms of area to be scanned as well as the detection of meteorological features, and these preferences must be incorporated into sensing operation. The differing sensing preferences of disparate end-users and reactive sensing strategies mean that it may not be possible to satisfy the needs of all the end-users and the required data generation time interval for particular radar algorithms at the same time.

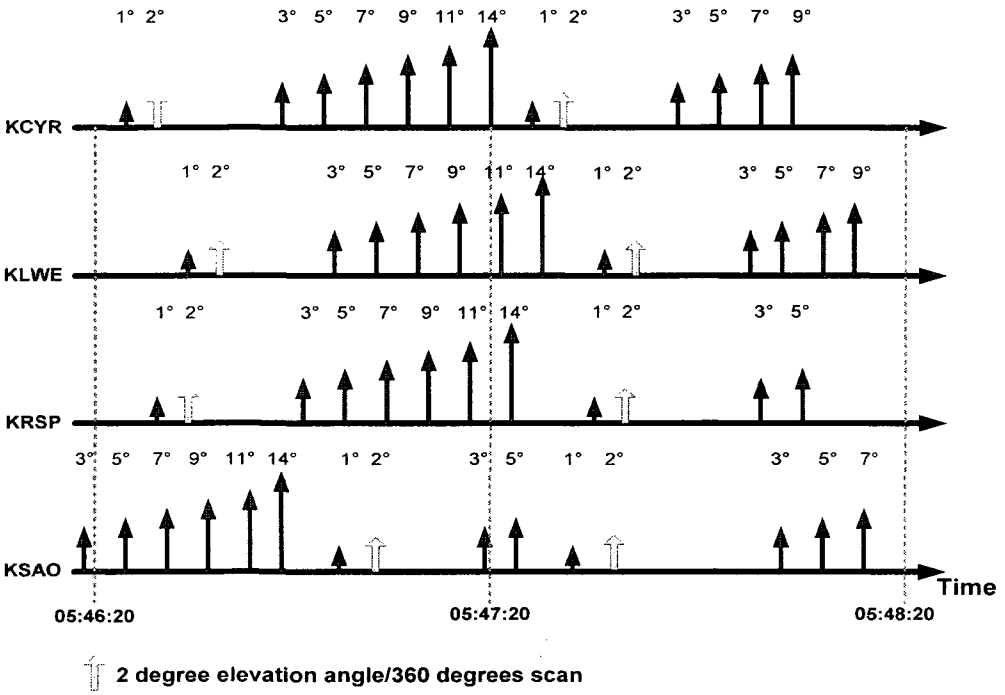


Figure 6.2 CASA IP-1 data generation timing

Furthermore, unlike the "sit-and-spin" mode of operation of traditional radars in which each radar continuously performs 360 degree scans, independent of the meteorological features present in the coverage area, each of the CASA radars can be dynamically assigned to scan an arbitrary starting point and at any width from 60 degrees to 360 degrees in the azimuth angle during a heartbeat interval based on detected meteorological feature location [Ku06]. This means a radar is assigned to scan 270 degree wide in azimuth angle at a particular elevation angle, and another radar scans only 90 degree wide in azimuth angle during the same heartbeat interval. Radar scan time depends on the size of assigned area, so each radar can take different amounts of time for scanning a particular elevation angle. Fig.6.2 illustrates an example of data generation timing of CASA IP1 testbed. The testbed is composed of a network of four radar nodes located at Chickasha (KSAO), Cyril (KCYL), Lawton (KLWE) and Rushsprings (KRSP), in Oklahoma [Br05]. The difference in the radar scan time for a particular elevation angle results in unsynchronized data generation times. These data generation times were collected in the testbed during a period of time when a storm cell

was moving through the coverage area of the testbed. Four radars (KSAO, KCYR, KLWE, and KRSP) in the testbed generate data in several sequential elevation angles within a minute. In Fig. 6.2, the different heights of vertical arrows represent the different elevation angles. The multi-radar data fusion algorithm considered in this chapter currently requires a set of radar reflectivity data captured at 2 degree elevation angle with full 360 degree scan in the azimuth angle. At each of 360 degree scan of the atmosphere, single radar generates about 1MB reflectivity data per minute. Furthermore, in order to properly track weather conditions, the algorithm needs to be run at least once in a minute. As seen in Fig. 6.2, each of the radars generates the required 360 degrees scanned radar measurements every 1 minute, however, the data generation time of the radars are not synchronized, thus we need to select a set of data considering the real-time requirement of the sensor applications and acceptable time difference among data to be integrated together. We will describe a data synchronization mechanism in Section 6.3.

6.1.2 Multi-radar Data Fusion Algorithm

Fig.6.1 illustrates the multi-radar data fusion procedure where data acquired or received from remote sensor nodes are used as supplementary information for correcting sensing errors in the data collected by the local radar. As illustrated in Fig 6.1, node A requires local data, gathered from radar A, as well as data from radars at nodes B and C to correct its sensing errors such as those due to attenuation of radar signal. Similarly node B needs the data collected by nodes A and C. In addition, the multi-sensor fusion algorithm usually requires a number of steps to obtain the final result. As illustrated in Fig. 6.1, multi-radar data fusion algorithms generally consist of two steps, preprocessing and main data fusion processing. In the first step, collected raw radar measurements are preprocessed or subsampled to meet the particular requirements of the individual applications.

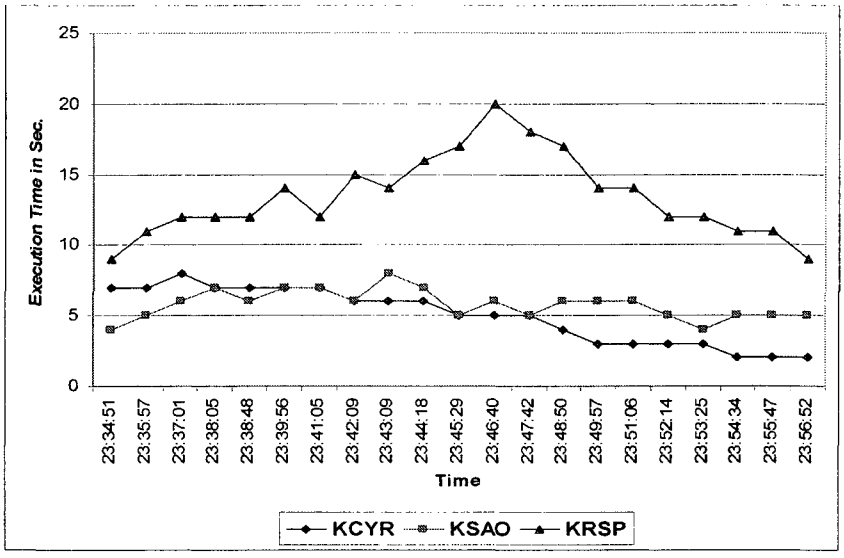


Figure 6.3 The transition of execution time of data fusion processing as a storm passed by the test-bed around midnight of April 10, 2007

In the second step, the algorithm detects significant features by integrating the preprocessed data. Due to large amount of the data collected by each of the radars, even this simple quality control, i.e., preprocessing step, of the data takes considerable time. We have implemented the multi-sensor data fusion algorithm on a machine (3.16GHz Intel CPU, 2GB RAM) running Linux [Lim07] using a client-server architecture. Fig. 6.3 shows the execution runtimes for per-sweep data processing including preprocessing and data fusion processing. We measured the execution runtime for multi-radar fusion processing in a case of severe thunderstorm that developed across CASA IP1 test-bed on the night of April 10, 2007. Our measurements show that the average execution runtime for merging a set of radar sweeps of the meteorological event is approximately 10 seconds.

6.1.3 P2P organization

MC&C of the CASA network gathers weather data from the radars and runs weather detection algorithms and selects a set of weather-scanning tasks of interest. MC&C then determines a set of

radars for the weather-scanning tasks. In these scenarios, several nodes that collaborate on the same weather-scanning tasks form a data fusion group. Although human input is used to cluster data fusion groups, the primary process of clustering will be mostly automated. A data fusion group consists of less than tens of radar nodes in practice. Because the radar nodes in the same data fusion group exchange data item to be integrated, the placement of data items are not restricted to the radar nodes that generated the data items. In order to locate the required data items or computing/communication resources a peer queries its neighbors that are members of the same data fusion group using a broadcasting-based peer probing mechanism. Since a relatively small number of radar nodes are involved in a data fusion group, such broadcasting-based peer probing mechanism does not impose a significant load in the network. Within the CASA network, the underlying network infrastructure may be subjected to adverse conditions due to severe weather, link degradation/outage, and variable cross-traffic along wired and wireless links. In such conditions, radar nodes can suddenly leave a data fusion group and return back to the data fusion group intermittently. This broadcasting-based mechanism is resilient to the dynamic node joins/leaves problem because each node in the data fusion group can be aware of a neighbor node's failure by querying all the neighbor nodes in the same data fusion group.

6.1.4 Problem Description

As previously mentioned, two processing steps are required to obtain the final multi-sensor fusion result: The first step (PP) is to preprocess individual data items from the individual radars to remove anomalies, and second step, fusion processing (FP) is to form more accurate meteorological feature estimates by merging the data items. Use of data from one radar to correct for attenuation encountered by a radar signal of another radar is a typical example of FP. Let $N = \{e_1, e_2, e_3 ... e_n\}$ be the given data set to be merged, and $S = \{s_1, s_2, s_3 ... s_m\}$ denote the set of sensor/processing nodes which are participating in a particular multi-radar data fusion algorithm.

Let $t_{PP}(e_i)$ be the estimated computation time of the PP step for data e_i, and $t_{FP}(e'_1, e'_2, e'_3 ... e'_n)$ denote the estimated computation time of FP using an intermediate data set $I = \{e'_1, e'_2, e'_3 ... e'_n\}$, where e'_i is the output of PP on the data e_i. Let $t_{in}(e_i)$ be the estimated data transfer time required to bring the data e_i to preprocess, and be $t_{out}(e'_i)$ the estimated data transfer time to send e'_i to the sensor node that is running FP. We assume that PP of individual data in the sets are independent each other, and the sensor nodes are allowed to conduct in-network processing, thus PP can be performed on multiple sensor/processing nodes concurrently. Assuming that FP starts immediately after receiving intermediate data set I, the total estimated processing time T for the multi-sensor data fusion algorithm is:

$$T = \max_{i \in N}[t_{in}(e_i) + t_{PP}(e_i) + t_{out}(e'_i)] + t_{FP}(e'_1, e'_2, e'_3 ... e'_n)$$

Our main goal is to integrate information from multiple sensors while reducing response time. In order to reduce the required time to obtain the data items, we assign a required data item e_i to a peer s^i_{pp} such that:

$$s^i_{pp} = \arg\min_{j \in S}[t_{in}(e_i) + t_{PP}(e_i) + t_{out}(e'_i)]$$

where argmin denotes the global minimum.

6.2 Node Architecture

In this section, we describe the software architecture at each node and identify key components required for multi-radar data fusion processing. Fig. 6.4 illustrates the software architecture at each node. Each node in the sensor network has an event handler, a peer manager and a resource manager. A multi-radar data fusion application needs to continuously collect and integrate data generated from a distributed group of radar nodes. There are a number of radar nodes exchanging

data to be integrated, while all the required data may not be available at the time of processing due to the difference in the generation time of a particular type of data as illustrated in Fig. 6.2. Therefore, the data fusion applications have to be notified of the availability of new data whenever new data are generated. To provide the notification service, the event handler implements three actions:

1) Delivering the data generation notifications to other nodes. A notification does not include data itself, but it contains the event source, data generation time, location, and size.
2) Providing a subscription mechanism to data fusion application. Each data fusion application is interested in only a particular type of data. Data fusion applications express their interests by registering a subscription with the event handler to be notified of any forth-coming events matching the subscription.
3) Informing data fusion applications of the generation of data by the local and remote sensors.

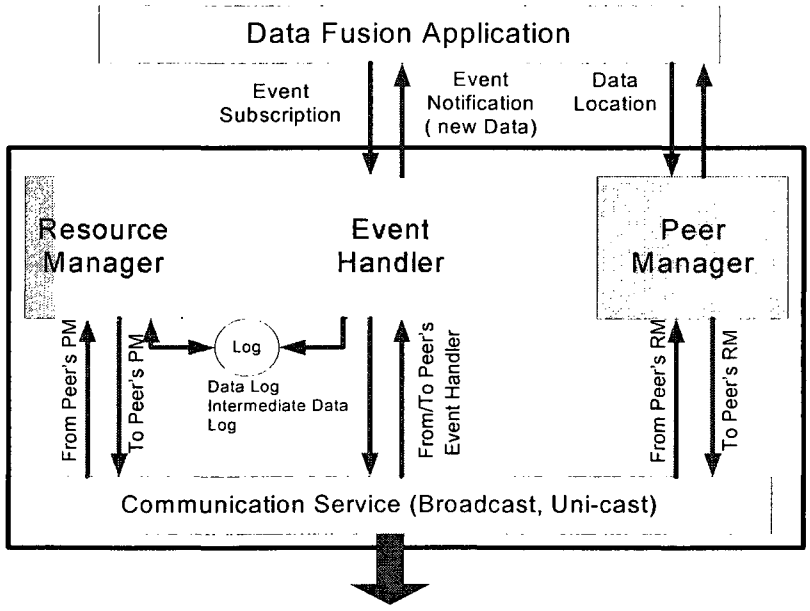


Figure 6.4 Node architecture

The resource manager maintains the following information at a node:

1) The list of application tasks the node can offer and the history of execution time for the application tasks.
2) The current available bandwidth for incoming/outgoing data transfer. Every time a node communicates with another node, the resource manager of the node keeps track of the Exponentially Weighted Moving Average (EWMA) of incoming/outgoing bandwidth based on the time (ΔT) required to transmit/receive data and the size (S) of the data as follows:

$$IN_{EWMA} = \alpha * IN_{EWMA} + (1-\alpha) * (S_{IN} / \Delta T_{IN}),$$
$$OUT_{EWMA} = \alpha * OUT_{EWMA} + (1-\alpha) * (S_{OUT} / \Delta T_{OUT}),$$

   To avoid being too heavily influenced by temporary bandwidth fluctuations, we currently set α to 0.8.
3) The log of new data generation events from local and remote nodes. In addition to the data generation events, the log also contains the information about the outputs of various processing (e.g., preprocessing output, and the final results of the algorithm) and replicated contents from other peers.

As mentioned before, the multi-radar data fusion algorithm requires obtaining data from other radar nodes. Obtaining data involves locating peers who have copies of the desired contents. Peers that have the desired contents are usually restricted to the peers who share same interest of processing. In order to achieve good performance for locating the contents, the peer manager maintains a list of peers who participate in the same multi-sensor data fusion processing.

6.3 Data Synchronization

When the event handler notifies the algorithm of a new data generation, the algorithm attempts to start the multi-radar data fusion processing. First step of processing is to search the event log to

find a proper set of data coming from other radar nodes to combine them with the local data. However, as seen in Fig. 6.2, the data generation times of the radar nodes are not synchronized; thus the algorithm may not be able to find a proper data set. Therefore, it is required to allow the algorithms to decide their data synchronization criterion in terms of the minimum number of data (N) from different radars for acceptable final result, the maximum possible waiting time for data (W) when the number of data is not sufficient, and the maximum tolerance in data generation time difference among the data to be integrated (D). Fig. 6.5 illustrates an example of data synchronization for multi-radar data integration algorithms. In the example, a multi-radar data fusion application merges data from three radar nodes A, B, and C with W=5, D = 10, and N = 3. At time t=16, node C generates a new data item C16 and the event handler running on node C posts the notification of available data. Then the algorithm running at the node C examines the event log to find whether data from other nodes are available. As seen in the figure, node A generated an appropriate data item A07 at time t=7, but node B has not generated any data in the past 10 seconds. Thus, the algorithm has to wait until the data selection criterion (N = 3) is satisfied. At t=20, node B generates a new data item, B20 and the difference 20 and 16 is less than W=5. Because all the criterion is satisfied, the algorithm can start processing at time t=20.

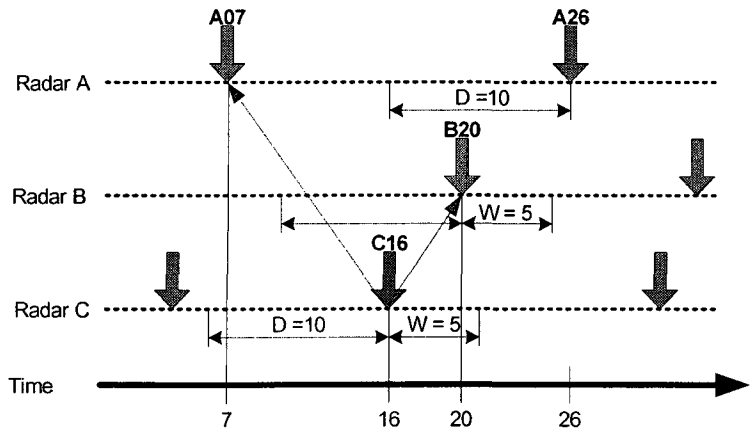
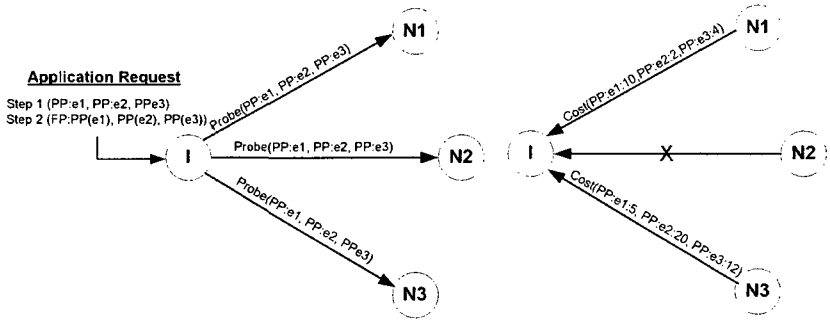


Figure 6.5 Data synchronization example

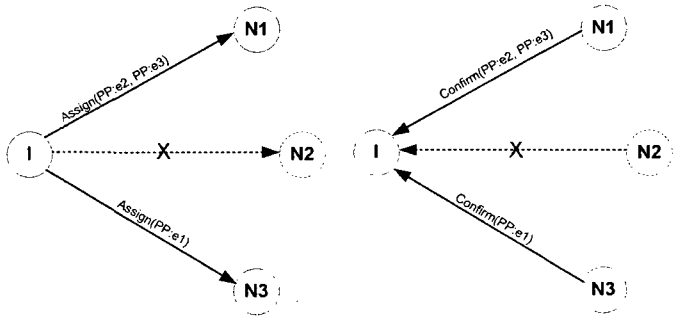
6.4 Best Peer Selection Protocol

After choosing a set of data to be merged during the data synchronization phase, a data fusion application starts data fusion processing by passing a request to BPS protocol. We use the term Initiator node for a node on which the application starts its data fusion. The peers that participate in the same multi-radar data fusion group with the Initiator node are defined as its Neighbor Peers. Initiator node's BPS protocol has four states - Initiator-Init, Initiator-Probe, Initiator-Assign, and Initiator-Wait. Likewise, each of the Neighbor peers has Neighbor-Init, Neighbor-Probe, and Neighbor-Proc states. Many of the multi-radar fusion applications require preprocessed data items to obtain the desired accuracy in the final fusion processing results. Because individual data items from multiple radar nodes can be preprocessed independently, preprocessing the individual data executes on multiple peers. In addition, the same set of data can be used by multiple executions of the algorithm on different nodes to correct sensing errors in the data collected by individual radars. Thus, a group of nodes interested in the same set of data can share intermediate outputs that are obtained by preprocessing instead of using raw radar measurements. To exploit those properties of multi-radar fusion processing, the application properly divides the entire process of the algorithm into several sub-processes. We assume that the entire process of the algorithm considered in this chapter is divided into two sub-processes, i.e., Step I for obtaining preprocessed data items to be integrated, and Step II for applying data fusion algorithm to detect significant features by integrating the preprocessed data items. After decomposing the entire process, the application passes down an application request that consists of these sub-processes. Each of the sub-processes is composed of a set of processing items, and each processing item is specified by a type of required processing and an input data set for processing. In Fig.6.5, a data fusion application at node C selects a set of data items (A07, B20 and C16) during the data synchronization phase. The multi-radar data fusion application requires obtaining the preprocessed data from the selected data set before fusion processing can begin.



(a) Initiator-Init State: Application request is passed down to BPS protocol; The Initiator broadcasts PROBE message that contains required processing.

(b) Initiator-Probe State: Neighbor nodes report the estimated response time for the processing items.



(c) Initiator-Assign State: The Initiator assigns the processing items to the peers on the active peer list.

(d) Initiator-Wait State: The Initiator waits for the completion of the processing items .

Figure 6.6 BPS protocol

Thus, Step I sub-process consists of 3 processing items and each of the processing items has a processing type (PP) and a data item as an input for processing. Step II sub-process has a single processing item, which consists of a processing type (FP) and the set of data items obtained by Step 1 sub-process. Fig. 6.6 illustrates how BPS supports the multi-radar data fusion algorithm for better performance. In the figure, the Initiator node waits for a request from a radar application in the Initiator-Init state. Upon receiving the application request, the Initiator node broadcasts a probing message for Step I, PROBE, to its neighbor peers (Fig. 6.6(a)). After broadcasting the PROBE message, the Initiator moves to the Initiator-Probe state, and waits for neighbor peers' response to the PROBE (Fig. 6.6(b)). The arrival of a PROBE message from the Initiator node causes the Neighbor peers to move from the Neighbor-Init state to the Neighbor-

Probe state. In the Neighbor-Probe state, the Neighbor peers calculate the estimated time for providing the processing items specified by the PROBE message. The details of the time estimation are described in the following subsection. After estimating the required time, each of the Neighbor peers responds to the Initiator node with a COST message, which contains the estimated time for providing the desired contents. As illustrated in Fig. 6.6(b), due to network congestion or outage or high system load, some peers may not be able to respond to the PROBE message in a certain time out period. In such cases, the Initiator node simply assumes that the Neighbor peers not responding are currently unavailable. Alternatively, the neighbor peers that can respond successfully before the time out period is expired are regarded as active peers. The pseudo-codes in Fig.6.7 describe the behavior of Initiator Node in *Initiator-Init* state. When the Initiator Node receives the COST messages from all the neighbor peers or the time out period is expired, the Initiator node moves from the *Initiator-Probe* state to the *Initiator-Assign* state.

```
1 Input: Item  items[];          /* processing items from application */
2 Input: int   peer_size;        /* the number of neighbors          */
3
4 Peer    activePeers = NULL;  /* active peer list              */
5 int     cnt = 0;             /* responding peer counter */
6 int     i;                   /* index variable                */
7
8 settimer(TimeOut);       /* start timer with the timeOut value*/
9 while ( timer_expire() != true && (cnt < peer_size ) ) {
10  /* receive COST message from neighbors */
11  Message mesg = recv_mesg(COST);
12  if (mesg != NULL) {
13    /* add the responded peer to the active peer list */
14    activePeers[cnt].peerID = mesg.peer;
15    for each items[] i {
16      activePeers[cnt].items[i] = items[i];
17      activePeers[cnt].items[i].cost = mesg.items[i].cost;
18    }
19    cnt++;
20  }
21}
```

Figure 6.7 Initiator-Init state algorithm

In the *Initiator-Assign* state the Initiator selects the peer among the active peers with the minimum estimated time for a particular processing item based on the COST message. Once the Initiator node selects the neighbor peers for all the processing items, it assigns processing items to the selected peers by sending ASSIGN messages (Fig. 6.6(c)). After assigning the processing items, the Initiator node moves to the *Initiator-Wait* state. In the *Initiator-Wait* state, the Initiator node waits for CONFIRM messages from the neighbor peers, which are notifications of the completion of the assigned processing items (Fig. 6.6(d)).

```
1  Input: Item   items[];     /* processing items from Initiator Node's mesg */
2
3  Message       mesg;       /* response message */
4  int           i, j;       /* index variables     */
5
6  for each items[] j {
7    mesg.items[j] = items[j];
8    mesg.items[j].cost = 0;
9
10   for each data i in items[j].dataset {
11     /* get data from log */
12     LOG log = get_log(items[j].dataset[i]);
13     switch(log.status)
14     {
15       case NotExist:
16         mesg.items[j].cost += tin(items[j].dataset[i])
17                               + tcomp(items[j].dataset[i]) + tout(items[j].dataset[i]);
18         break;
19       case ExistRaw:
20         mesg.items[j].cost += tcomp(items[j].dataset[i]) + tout(items[j].dataset[i]);
21         break;
22       case Exist:
23         mesg.items[j].cost += tout(items[j].dataset[i]);
24         break;
25     }
26   }
27 }
28 /* reply to the Initiator */
29 send_mesg(mesg, COST);
```

Figure 6.8 Neighbor-probe state algorithm

When an ASSIGN message arrives at a neighbor peer, the Neighbor peer moves to the neighbor processing state, *Neighbor-Proc*, and provides the desired contents. After providing the desired contents, the neighbor peers notify the Initiator node of the completion of the assigned processing items with the CONFIRM messages. Upon receiving a CONFIRM message from a neighbor peer, the Initiator node retrieves the item that is assigned to the neighbor peer, and starts processing FP.

A. Neighbor Peer Probing

In order to respond to the Initiator node's PROBE message, the neighbor peers calculate the communication and computation costs using simple estimation schemes. In BPS, peers do not use expensive bandwidth and processing power estimation tools to determine communication and computation costs precisely, although future integration of such tools is possible. Instead, the following simplistic approach is used for cost estimation: the resource manager of each peer keeps track of the EWMA of incoming/outgoing throughput as mentioned in Section 4. Similarly, the resource manager also maintains the history of the computation time for particular types of processing. The computation time, $t_{comp}(item[i])$ for the *item[i]* can be decided by the recent computation time for the same type of processing in the history list. Likewise, we define the estimated data transfer time to acquire data *e*, and estimated data transfer time to send *e* after the processing as $t_{in}(e)=e.size/IN_{EWMA}$, and $t_{out}(e)=e.size/OUT_{EWMA}$, respectively. The cost estimation algorithm is illustrated in Fig.6.8. The first step to the cost estimation is to determine the status of processing items. In order to determine the status of a processing item, the peer node reads information about the processing item (Recall from our discussion in Section 4 that each node has a log to store radar data generation events and the information about the outputs of various processing.). A processing item can have one of the following statuses: *Exist, ExistRaw* and *NotExist.* If the status of a processing item is *"Exist"* at a neighbor peer, then the neighbor has the output of required processing, so the Initiator node needs only the cost of communication to obtain the processing item from the neighbor node. *"ExistRaw"* means that only the raw data

exists at the neighbor; therefore, the cost of computation for required processing and the cost of communication are needed. In the case of the *"NotExist"*, the neighbor node has neither the raw data nor the output of the processing item, so the neighbor node needs to bring the raw data from other peers and to run required processing. Therefore, in many cases, it is not likely to assign the processing item to a neighbor node when its status of processing item is *"NotExist"*.

B. Processing Item Allocation

The cost information provided by the neighbor peers is used to assign the processing items to the active peers. Fig. 6.9 illustrates Initiator-assign state algorithm. During the iterations of the assigning procedure, from the unassigned processing items a new processing item is selected and assigned to an active peer that has the minimum cost for the selected processing item. After assigning the processing item to the peer, BPS protocol adds the cost of the selected processing item to the other processing items' costs estimated by the selected peer.

```
1 Input: Item  items[];            /* processing items from application */
2 Input: Peer  activePeers;        /* obtained from probe state        */
3
4 Peer     peer;
5 int      i, j;                    /* index variables */
6
7 for each items[] i {
8    /* find peer which has minimum cost for item i */
9    peer = find_min_cost_peer(activePeers, items[i]);
10   /* assign item i to this peer  */
11   peer_assign(peer, items[i]);
12   /* update cost of the selected peer */
13   for each items[] j {
14       if ( items[i] != items[j] ) {
15          peer.items[j].cost += peer.items[i].cost;
16       }
17   }
18 }
```

Figure 6.9 Initiator-assign state algorithm

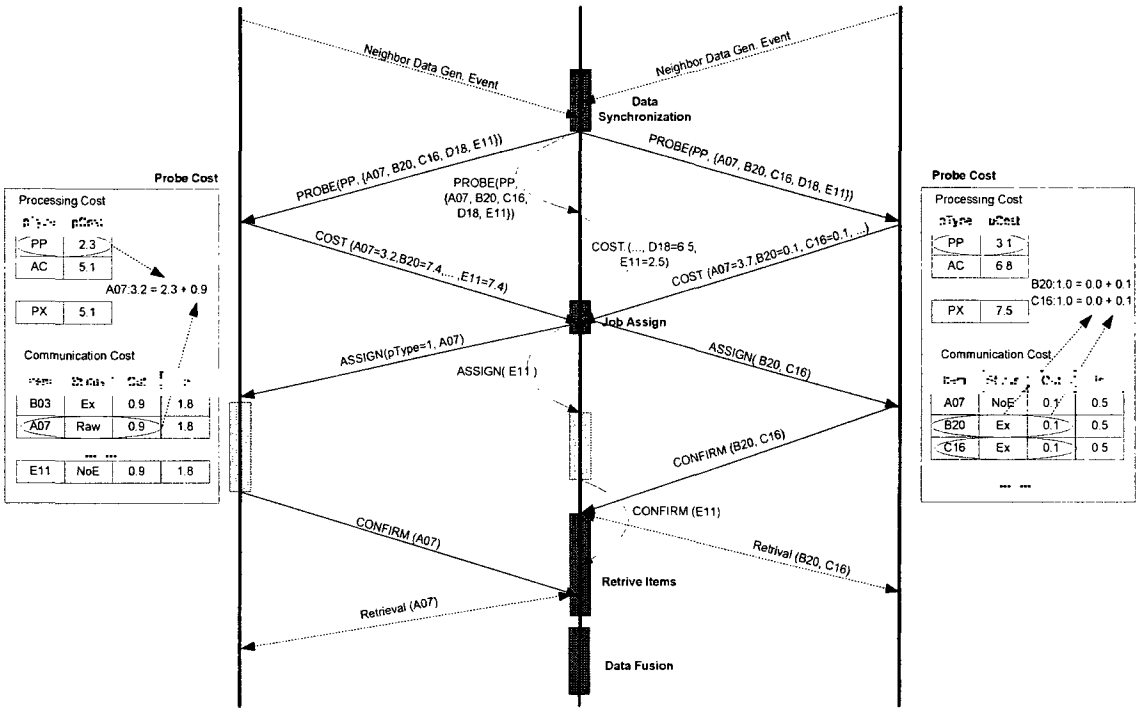


Figure 6.10 Example of BPS protocol operation

By updating the other items' estimated cost, the chance for assigning multiple items to the same peer can decrease. The process is repeated until all processing items are assigned to the active peers.

C. An Example

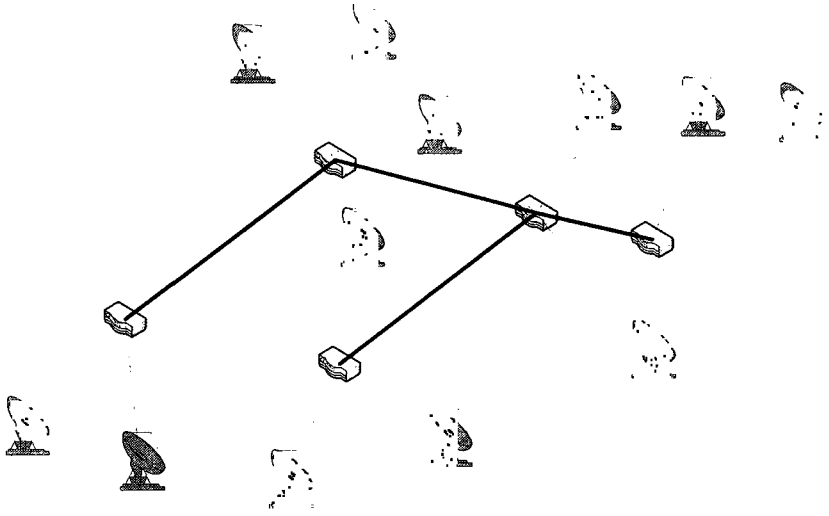
Fig. 6.10 illustrates how an Initiator node distributes required processing items to its Neighbor peers using BPS algorithm. During the data synchronization phase, a multi-sensor data fusion application chooses a set of data items. Each of the peers in the network delivers a data generation event to neighbor peers whenever it captures data from the physical environment. Based on the data generation notification, a multi-radar data fusion can select a set of correlated data items to be combined. Because a notification does not contain the captured data item itself, it is required to obtain the data items from Neighbor peers. As mentioned earlier, the peer manager of Initiator node broadcasts a probing message to its Neighbor peers' resource manager in order to obtain the

selected data items. Due to the fact that the peer manager at each node regards itself as one of its Neighbor peer, the probe message is also delivered to the resource manager of the Initiator node when the Initiator node sends out the probe message for the required data items. In this example, we assume that 5 data items, i.e., A07, B20, C16, D18, and E11, are selected by the data fusion application. By probing its Neighbor peers, the Initiator node assigns data item A07 to Neighbor peer 1, B20 and C16 to Neighbor peer 2, E11 to itself and D20 to another neighbor node, respectively. Neighbor peer 2 has data items B20 and C16, and both B20 and C16 have been preprocessed before. Thus, Neighbor peer 2 can immediately respond to the ASSIGN message with a CONFIRM message. As depicted in the Fig. 6.10, Neighbor peer 1 has data item A07, but the data format is a raw radar measurement. Thus, Neighbor peer 1 requires preprocessing data item A07 upon receiving an ASSIGN message from the Initiator node. After preprocessing A07, Neighbor peer 1 replies to the Initiator node with a CONFIRM message as the notification for the completion of the assigned processing item. The multi-sensor data fusion applications require the availability of all the data items before the main fusion processing, FP, can begin. Thus, the Initiator node collects a data item from a neighbor node when it receives a CONFIRM message, and starts FP after collecting all the required items from the neighbor nodes.

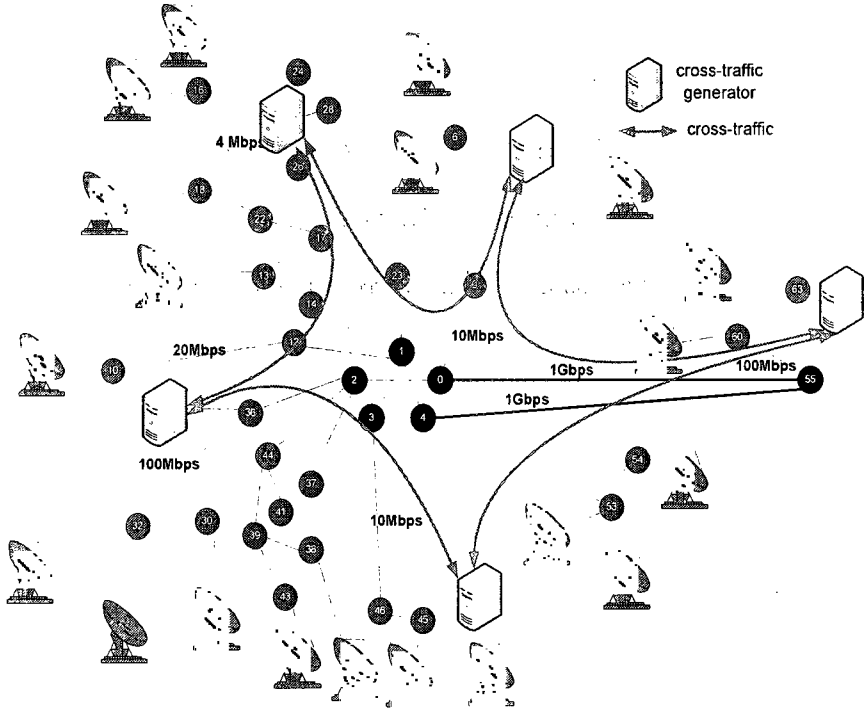
6.5 Experimental Results

CASA IP-1 testbed [Br05], with its four nodes, is a very limited environment for evaluating the scalability and performance of the data-fusion scheme. Therefore, we present results for larger networks based on SimGrid[Le03], a discrete time simulator. The parameters, when applicable, were based on testbed or measured values. The sensor nodes in our simulation sensor networks were divided into groups of sensor nodes for the multi-radar data fusion. As described in Section 5, an application specifies a synchronization preference, such as the minimum number of data

(N), the maximum waiting time for data when the number of data items is not enough (W), and the maximum tolerance in sampling time gap among the data to be integrated (D). For the experiments, D=45 sec., W=5 sec., and N=3, which correspond to typical radar meteorological application. During each minute, each sensor collected 1~2 new radar sweeps in a particular scan mode, and each data item size was about 1MB. We set the execution time for preprocessing on a sweep to 1 second. The processing times for the main data fusion are exponentially distributed with a mean of 3 seconds, and the maximum and minimum processing times of this step are 1 second and 12 seconds respectively. The metric we use for evaluating the performance of BPS is the response time, defined as the duration from the start of processing to the end of processing including preprocessing and main fusion processing. Thus, lower response times represent better performance. For the comparison, we also implemented two common heuristic algorithms: random and fixed peer selection algorithms. In the random algorithm, a set of peers was randomly chosen for obtaining data items without considering the resources or contents availability at the peers. In the fixed algorithm, the peers which generated particular data were always selected for obtaining the data items. The goal of BPS is to achieve better performance by finding a set of peers that can provide best services for the required processing. We conduct two sets of experiments to evaluate how well the BPS algorithm meets the goal. In the first set of experiments, we study the performance of BPS algorithm when the CPU availability was varied across the peers. The second set of experiments investigates the BPS algorithm's ability to accomplish better performance in the presence of variable cross traffic in the sensor network. Fig. 6.11 is the two network topologies that are used for the experiments. A number of experiments were conducted under variable CPU availability conditions across the sensor nodes using the network shown in Fig. 6.11(a). The network shown in Fig. 6.11(b) was used for estimating the performance of the heuristics under variable network load conditions.



(a)



(b)

Figure 6.11 Experiment topologies for the performance of heuristics under variable CPU availability conditions (a), and different network load conditions (b)

A. System Load Variation

First set of experiments was based on a network of 12 sensor nodes (Fig. 6.11(a)). In order to simulate the CPU availability variation among the peers, we selected half of the sensors randomly, and set CPU availability to average 20%, 50% and 80% on the selected peers. The other half of the peers were assigned average 95% of CPU availability. The 50% CPU availability means that the peer can deliver only the half of its computing power to the multi-sensor data fusion algorithm. The end-to-end network bandwidth between any two sensor nodes is set to 100Mbps, thus the communication overhead for the processing is small in this set of experiments. Fig. 12 shows the simulation results for the first set of experiments. In Fig 12(a), that characterizes average response time vs. CPU resource availability, of the multi-radar data fusion algorithm, BPS clearly shows better results for all the conditions. The reason is that BPS selects peers for assigning processing items considering the peers' computation time history for the particular processing items but random and fixed do not. BPS algorithm eagerly attempts to find lower-loaded peers, which can finish the required processing earlier than higher-loaded peers, and assign the processing items to the peers. Fig. 12(b), Fig. 12(c) and Fig. 12(d), show the cumulative distributions of the response times of multi-radar data fusion algorithm. Compared to the fixed peer selection algorithm, BPS provides more consistent performance. As seen in the figures, even the half of peers is high loaded, almost 80% of multi-sensor data fusion algorithm response times are less than 20 seconds.

B. Network Load Variation

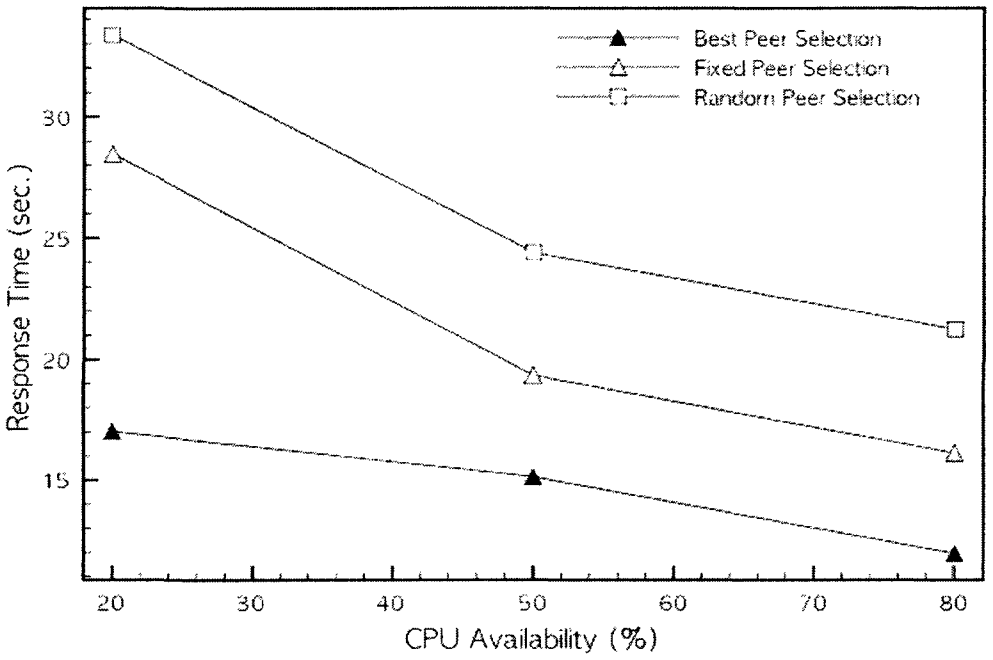
Next we investigate the impact of network load variation on the performance of BPS using a sensor network with 20 sensor nodes (Fig.6.11(b)). We used GT-ITM [Ca07] to create a 100-node transit-stub graph as our underlying network topology. The bandwidths on the network links were randomly assigned as 1Gbps, 100Mbps, 20Mbps, 10Mbps and 4Mbps, and the network latency were set randomly to 0.5ms ~ 5ms. Dynamic network conditions were created by locating network load generators in the sensor network.

	Best Peer Selection Response Time	Fixed Peer Selection Response Time	Random Peer Selection Response Time
20% CPU Availability	17.02 sec.	28.47 sec.	33.36 sec.
50% CPU Availability	15.14 sec.	19.34 sec.	24.41 sec.
80% CPU Availability	11.95 sec.	16.12 sec.	21.24 sec.

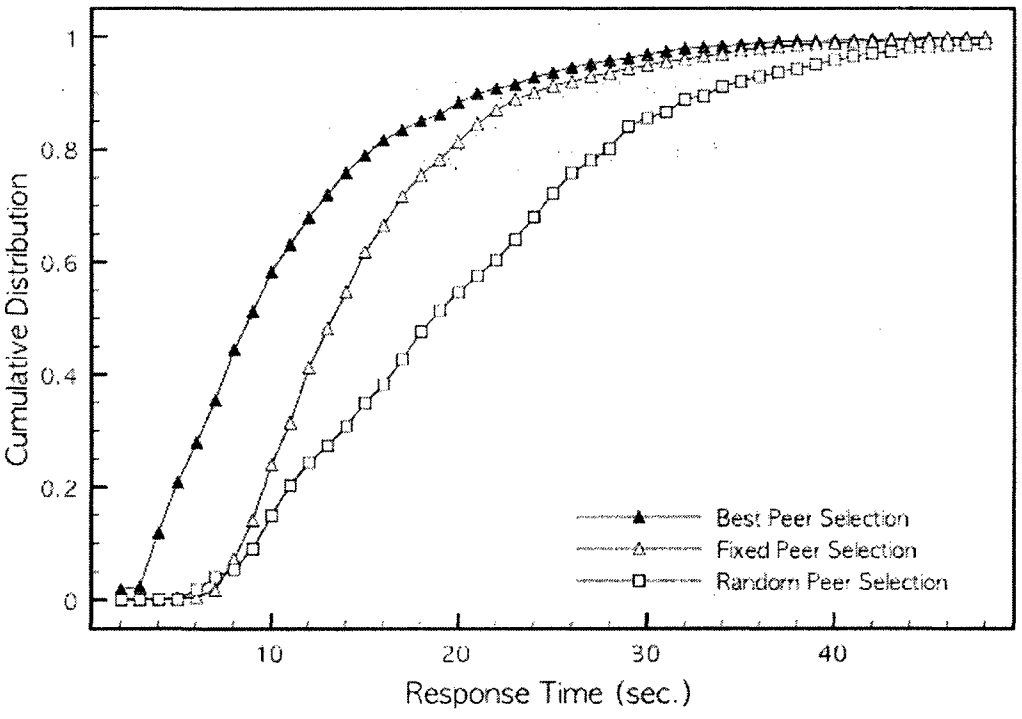
Table 6.1 Average response time under different system load

	Best Peer Selection Response Time	Fixed Peer Selection Response Time	Random Peer Selection Response Time
10% Cross-traffic	8.39 sec.	10.77 sec.	18.75 sec.
20% Cross-traffic	8.89 sec.	11.99 sec.	20.73 sec.
30% Cross-traffic	10.19 sec.	13.69 sec.	23.70 sec.
40% Cross-traffic	11.06 sec.	15.13 sec.	27.38 sec.
50% Cross-traffic	11.71 sec.	17.71 sec.	35.46 sec.
60% Cross-traffic	14.31 sec.	21.52 sec.	41.95 sec.
70% Cross-traffic	15.52 sec.	26.92 sec.	57.69 sec.
80% Cross-traffic	27.28 sec.	39.20 sec.	69.25 sec.
90% Cross-traffic	30.24 sec.	52.03 sec.	75.54 sec.

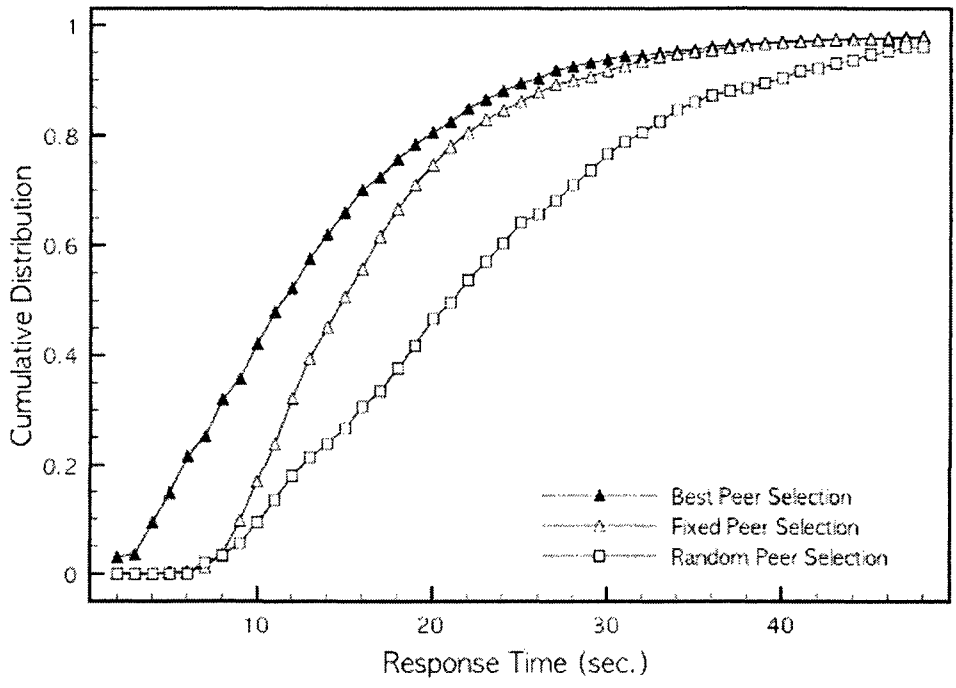
Table 6.2 Average response time different network cross-traffic conditions


Best Peer Selection
Fixed Peer Selection
Random Peer Selection
Response Time (sec.)
30
25
20
15
20
30
40
50
60
70
80
CPU Availability (%)

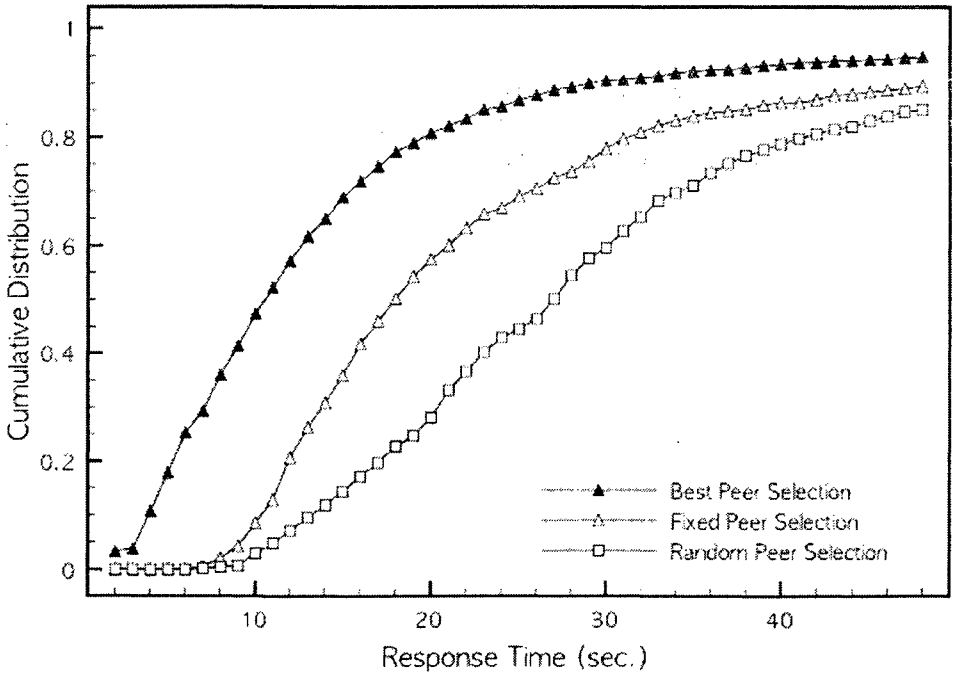
(a)


Cumulative Distribution
1
0.8
0.6
0.4
0.2
0
10
20
30
40
Response Time (sec.)
Best Peer Selection
Fixed Peer Selection
Random Peer Selection

(b)



(c)



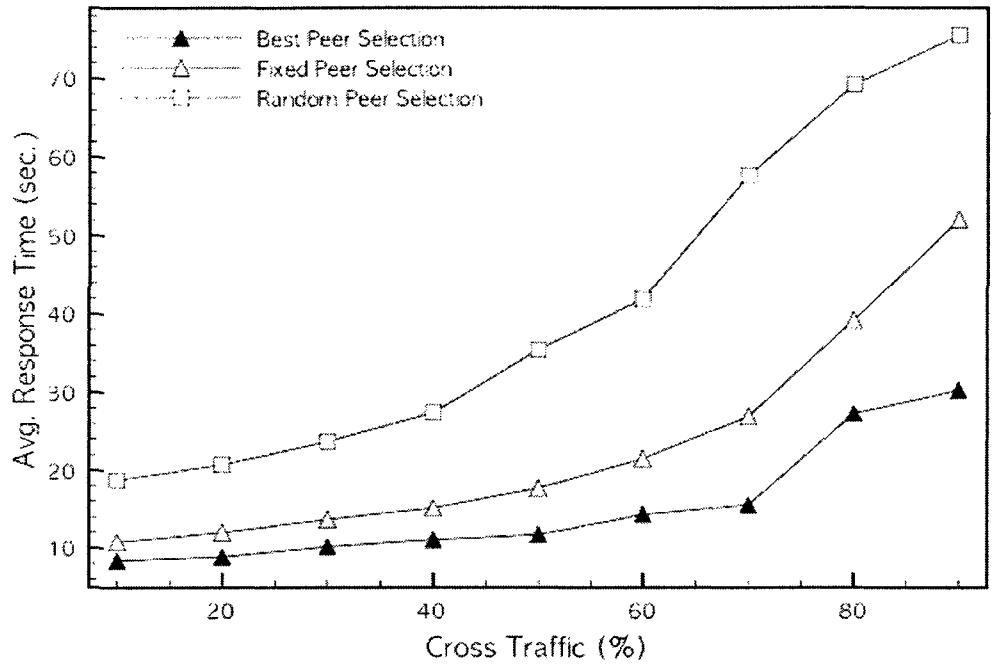
(d)

Figure 6.12(a) average response time under different system load, and CDF of response time under (b) 80%, (c) 50%, and (d) 20% CPU availability.

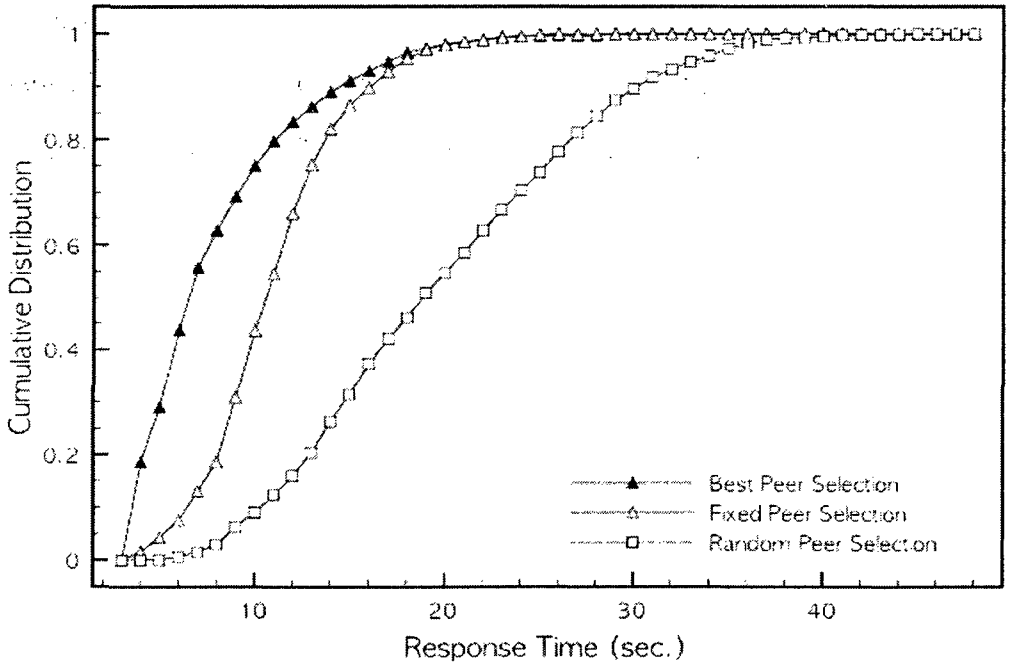
The load generators injected cross-traffic on shared communication links. We varied the ratio of the average cross-traffic (μ) to physical bandwidth from 0.1 to 0.9 to investigate the effect of network congestion on the response time. Plentiful evidence suggests that the network traffic is extremely variable and bursty [Le94][Pax06]. The burstiness was taken into account by generating the cross-traffic according to a type of self-similar process, *Fractional Gaussian Noise* (FGN) [Pax97]. The burstiness of network traffic is characterized by a self-similarity parameter (Hurst Parameter, H). For the simulations, we fixed H = 0.79, and the output variance = 0.3*μ. Fig. 6.13(a) shows the average response time achieved by BPS, fixed, and random under different amount of cross-traffic. The results show that BPS outperformed the other two heuristics under all network conditions. This is because the BPS algorithm always chooses those peers which can provide the desired contents with the minimum cost, while the fixed peer selection heuristic always gets the desired contents from the source peer of the contents. While collaborating on a multi-radar data fusion algorithm, data from a bandwidth-poor sensor node can be replicated by a bandwidth-rich peer. Once the replication is done, other peers can retrieve the copy of the desired contents from the bandwidth-rich peer in BPS instead from the bandwidth-poor source node. Fig. 6.13(b), Fig. 6.13(c) and Fig. 6.13(d) plots the CDF of response time that BPS, fixed and random heuristics obtain. Table 6.2 shows the data corresponding to the results shown in Fig.6.13 (a). Compared to the other heuristics, the majority of the response times are smaller than that of the other heuristics.

6.6 Summary

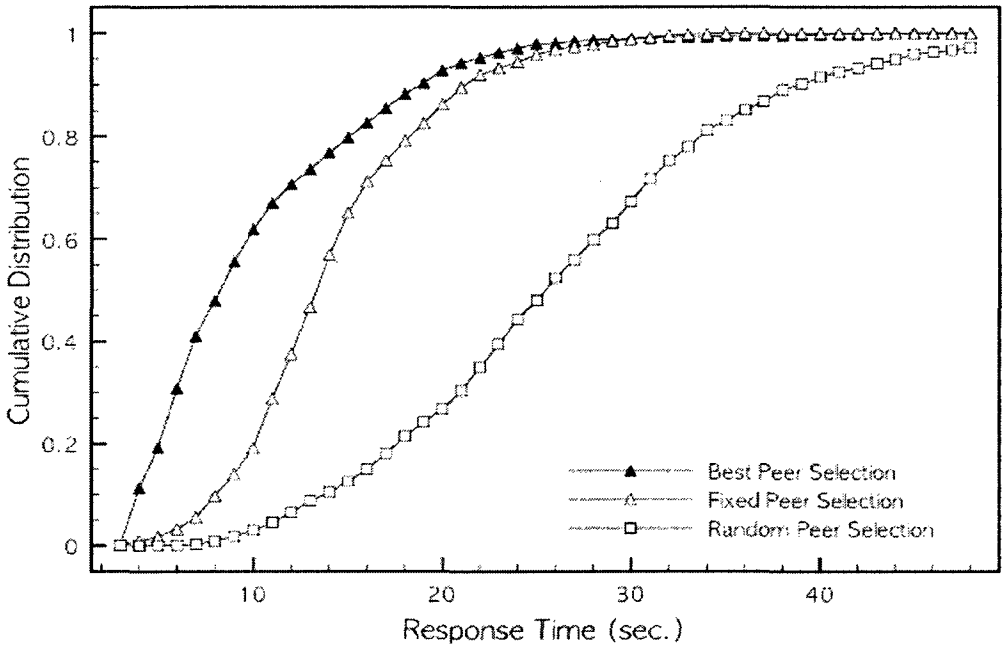
A P2P collaboration framework for multi-sensor data fusion in resource-rich radar sensor networks was presented. We proposed a data synchronization mechanism and a peer selection scheme to coordinate peers for multi-radar data fusion applications.


Best Peer Selection
Fixed Peer Selection
Random Peer Selection
Avg. Response Time (sec.)
10
20
30
40
50
60
70
20
40
60
80
Cross Traffic (%)

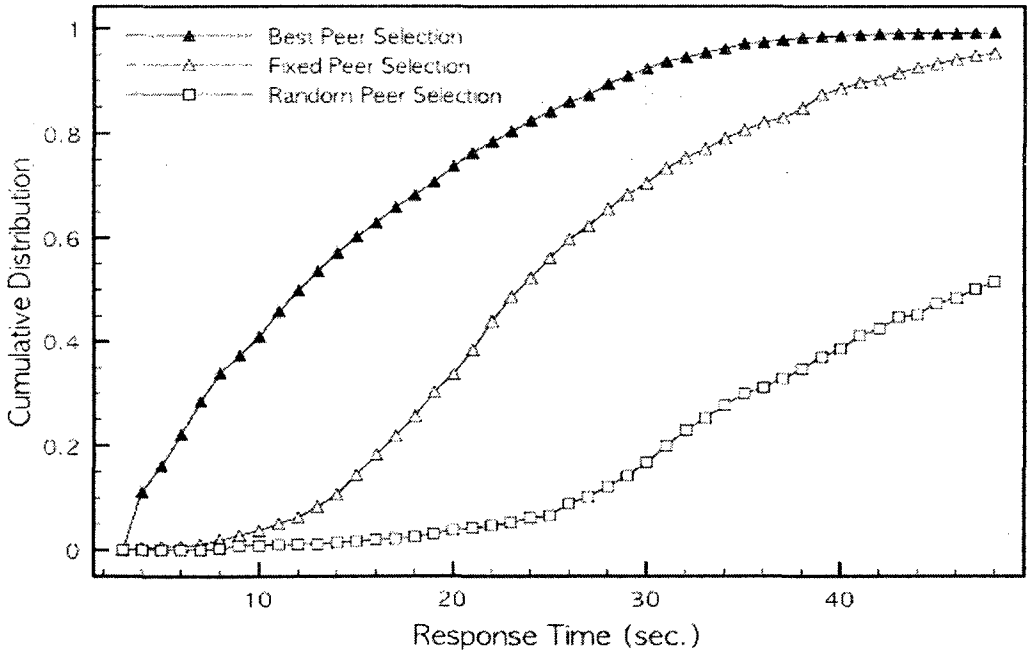
(a)


Cumulative Distribution
0
0.2
0.4
0.6
0.8
1
10
20
30
40
Response Time (sec.)
Best Peer Selection
Fixed Peer Selection
Random Peer Selection

(b)



(c)



(d)

Figure 6.13 (a) average response time under different network cross traffic conditions, CDF of response time under (b) 10%, (c) 40%, and (d) 90% network cross-traffic.

Our results illustrated the effectiveness of the proposed framework and heuristic algorithm. P2P networks could provide an effective framework for the deployment of such real-time multi-radar data fusion services as well as other emerging high-bandwidth real-time applications. On-going work includes a prototype of the framework that is being implemented in a real test-bed environment. We also plan to investigate how to include other desirable feature such as fault tolerance into our framework.

Chapter 7

AN ANALYTICAL MODEL FOR DATA FUSION LATENCY IN INTERNET-BASED SENSOR NETWORKS

Advances in sensor technology and computer networks have enabled the development of various types of sensor networks. These systems use a large numbers of distributed sensor nodes which sense, process and transmit the data revolutionizing information gathering and processing over wide area and in many situations. Because of its flexibility and global accessibility the Internet is used as the preferred medium for communication among the sensors and various end-users within same or different high-speed sensor networks [Mc05][Qi01][To04]. Some of the sensor networks are composed of small clusters of powerful sensors and the nodes of the sensor networks are directly connected to the Internet like personal computers or servers [Mc05][Zi05][Br05]. In addition, large swarms of micro-sensors which are wirelessly connected permit remote object monitoring and tracking in many different contexts. Research efforts in wireless sensor networks (WSN) have been mainly focused on the internal issues, such as routing, self-organization, MAC layer design, and collaborative data processing [Ak02][In00]. Recently, some research efforts focus on the interconnection of WSN's with the Internet using a gateway placed between a wireless sensor network and the Internet to provide value-added services through the interaction with a large number of remote users and service centers [Hw03][Mc05][To04]. Applications include distributed instrumentation, weather monitoring [Mc05], industrial automation, and structure health monitoring [To04]. These Internet-based sensor network applications often create

a closed-control loop; the loop begins with continuous data capture via sensors, followed by multiple stages of processing, and ends with actuators [Mc05]. At the lowest level, an individual sensor node or a group of sensors periodically takes raw measurements from physical environments. Initial data processing on the raw measurement can be carried out at the local sensor node. These raw measurements or initial processing results have to be processed at a processing center to enable higher-level inferences in order to make decisions or gain knowledge based on the information from distributed sensor inputs [Qi01a]. Data from multiple sensors can be used to extend the field of view and to increase the certainty and accuracy of sensor estimates. In this context, multi-sensor data fusion is an common and essential part of the sensor applications. Furthermore, in many of the applications data fusion processing should be completed within a certain time bound in order to guarantee the proper operation of the systems. However, in a best-effort network environment, timely data delivery is a challenging task because of dynamic resource fluctuations within the network. Previous measurements studies on network traffic in local area networks [Le94] and wide area networks [Pax95] show that the network traffic shows long-range dependence (LRD) or self-similarity. It is also known that the self-similar network traffic can have a detrimental impact on network performance, including amplified queuing delay and packet loss rate [Fr03][He96][Man02][No94][Pa00]. Furthermore, real Internet paths are different in available bandwidth, burstiness and other network characteristics, so they also differ in network delay. Each of the sensor nodes may be connected to different networks or communication links, and fusion processing typically requires availability of all inputs before the application of the fusion function. Therefore, these differences in network delay cause an extra synchronization delay for multi-sensor data fusion. As many of the high-speed sensor applications access sensor readings remotely via the Internet, and their relatively high data rate also contribute to the network delay, it is imperative to understand how the time-scale invariant network cross-traffic affects the performance of the sensor applications. In this chapter, we develop an analytical model for multi-sensor data fusion latency considering

most of the major aspects of the multi-sensor data fusion in Internet-based sensor applications. In order to study the impact of self-similarity on the multi-sensor data fusion latency, we model the cross-traffic using a Fractional Brownian Motion (FBM) process. The FBM is a strictly self-similar process which models the amount of work generated in any given time interval with a Gaussian distribution [No94][No95]. According to the Central Limit Theorem, the high degree of multiplexing of traffic justifies modeling the traffic aggregate as a Gaussian process [No94]. Numerous empirical evidences show that the model is valid for describing the self-similar network traffic characteristics [Fr03][Man02]. In addition, various studies have been done using the FBM model as a reference point for provisioning network bandwidth to satisfy certain QoS requirements [Fr03][No95]. Many of the sensors take measurements from physical environments in a periodic manner. For analyzing the multi-sensor data fusion latency, it is required to consider the periodic nature of sensor data generation and the network conditions which exhibit the self-similarity. We develop a Probabilistic Estimation of Periodic Backlog (PEPB) technique. Using the PEPB technique, we first estimate single-hop delay in the sensor network. Based on the single-hop delay estimation, we develop a method to compute multi-sensor data fusion latency in a sensor network. We envision wide applications of this approach. This fusion latency estimation technique can be used for provisioning network bandwidth and developing data synchronization strategies for multi-sensor data fusion applications. The remainder of the chapter is organized as follows. In Section 7.1, we identify the components of the end-to-end fusion latency. Section 7.2 introduce the FBM model and discuss its parameters. In Section 7.3, we present a probabilistic estimation of periodic backlog (PEPB) technique based on the FBM model. Section 7.4 presents a single-hop performance analysis using the PEPB technique. In Section 7.5, we extend our analysis two-hop sensor network with multiple sources. Section 7.6 concludes and discusses areas of future research.

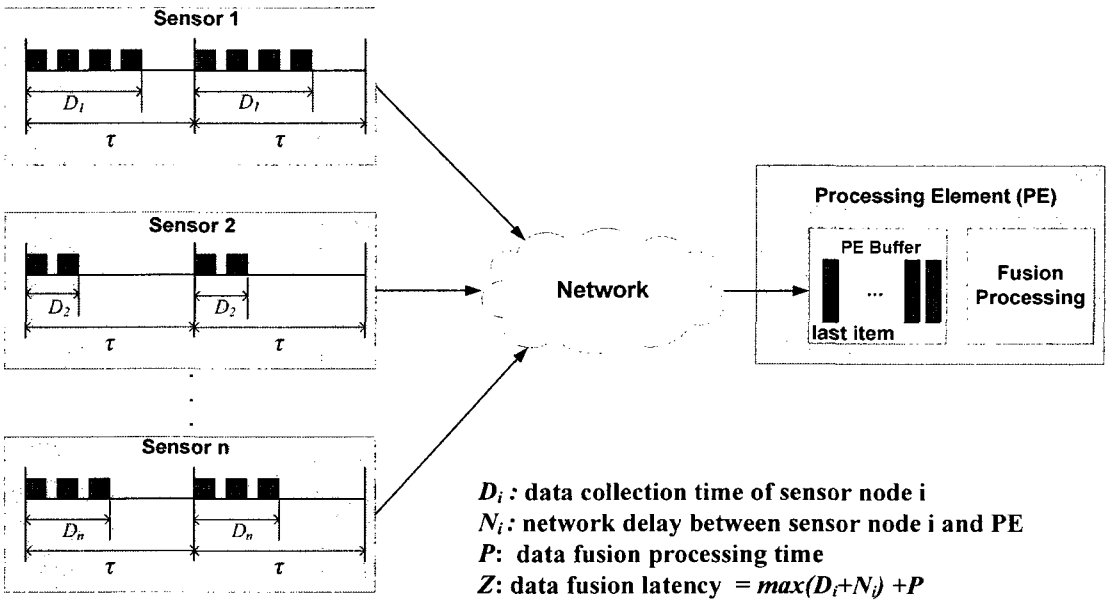


Figure 7.1 A sensor network model

7.1 Target System Model

Fig. 7.1 shows a sensor network model for multi-sensor data fusion. A sensor network consists of a set of sensor nodes, a processing element (PE), and a communication network interconnecting the sensor nodes and the PE. Collaborative Adaptive Sensing of the Atmosphere (CASA) [Mc05] is an example of such sensor networks. CASA is based on the concept of a network of small weather radars, integrated with a distributed processing and storage infrastructure in a closed loop system to monitor lower troposphere for atmospheric hazards like tornados, hailstorm etc. In this sensor network, sensor nodes periodically capture physical world phenomena every 'τ' time unit as shown in the figure. All the sensor nodes have a common period 'τ', but individual sensors can be assigned to a certain sensing task during the same time interval based on detected feature in the sensing area and user preferences. Sensor data acquisition time depends on the size of assigned area or sensing task, so each sensor node can take different amounts of time to complete

the assigned sensing task. However, it is assumed that the beginning of the data acquisition is synchronized for all the sensor nodes.

After collecting data, a sensor node transfers the data through the communication network to the PE, where fusion processing of data from different sensors takes place. The communication network is shared by the sensor nodes and other traffic sources. In the communication network, scale-invariant self-similar cross-traffic can affect the network performance. Data fusion processing requires the collection or aggregation of correlated data items, and data correlation is typically time-based, requiring aggregation of items generated within the same time interval. In this sensor network model, we assume that the PE requires availability of all the items before starting fusion processing. We define the fusion latency as the duration from the beginning of data acquisition to the end of fusion processing. Therefore, the fusion latency can be modeled with four high-level components:

1) Data acquisition time (D) is introduced at individual sensor nodes. In some sensor applications, each sensor node takes a raw measurement from a physical environment, and pre-processes the raw measurement. For simplicity, we assume that data acquisition time includes both the time for taking raw measurement and the time for preprocessing.
2) Network delay (N) is the time need to transfer data from the sensor nodes to the PE. As data travels from one of the sensor nodes to the PE through the communication network, the data suffers from several types of delays at intermediate network nodes along the path. The delays are propagation delay, transmission delay, nodal processing delay and queuing delay. Since the nodal processing delay and propagation delay can be neglected, we take queuing delay and transmission delay into account for delay analysis.

$A(t)$	Traffic amount during time [0, t], $A(t) = mt + \sqrt{am}\,Z(t)$
m	Mean input rate
a	variance coefficient
C	constant service rate
H	self-similarity parameter, $0.5 \le H < 1$
$Z(t)$	normalized fractional Brownian Motion ($\mu=0$, $\sigma=t^H$)
$Z(t)$	standard normal PDF
$p(x)$	probability that the fusion latency is larger than x
$\Phi(a)$	standard normal CDF
s	sensor data generation rate
ρ	network utilization

Table 7.1 List of parameters

3) Data synchronization delay (S) is introduced at the PE. Because the PE starts fusion processing after all the data items arrive, extra time is needed up to the maximum network delay between the sensor nodes and the PE.

4) Processing delay (P) is the time required to process sensor data by applying data fusion algorithm at the PE.

If we use the beginning of the data acquisition as the reference time for the Fusion Latency (L_F), the fusion latency is:

$$L_F = \max_{i \in sensors} (D_i + N_i) + P \qquad (7.1)$$

where D_i, N_i, and P denote the data acquisition time at sensor node i, the network delay from sensor node i to the PE, and the time taken by fusion processing at the PE, respectively.

A typical performance requirement in real-time applications such as real-time multimedia applications would be the probability that the end-to-end delay exceeds a certain level [Fr03]. For the real-time sensor applications, it is also a reasonable method to specify the performance requirement in terms of a probabilistic delay requirement of the form

$$P\{L_F > z\} < \varepsilon, \qquad (7.2)$$

that is the probability that the total multi-sensor data fusion latency L_F exceeds a certain time z is less than ε. In this chapter, we measure the tail probability of the multi-sensor data fusion latency.

7.2 Probabilistic Estimation of Periodic Backlog

In Section 2.7, we discussed a queue length distribution with fractional Brownian motion input. The supremum of queue length of (2.5) can be expressed as

$$\begin{aligned} \{Q > x\} &= \sup_{t \geq 0}(A(t) - Ct) > x) \\ &= \bigcup_{f>0} \{(A(f) - Cf) > x\} \end{aligned} \qquad (7.3)$$

The union of (7.3) represents events when the queue length is larger than x at intermediate points in time f during the time interval [0, t). In other words, the supremum of queue length expression in (7.3) concerns whether there exist at least one event that the queue length exceeds a certain value x or not during the time interval. However, for our application we need to know the occurrence of the event when the sensor packet reaches the queue in order to model the queuing delay experienced by the sensor packet. Thus, the supremum distribution of queue length in (7.3) may not be able to capture the queuing behavior of the sensor applications.

In order to examine the queuing behavior of the sensor applications, let us now consider a network link with capacity C that is shared by the sensor(s) and other cross-traffic.

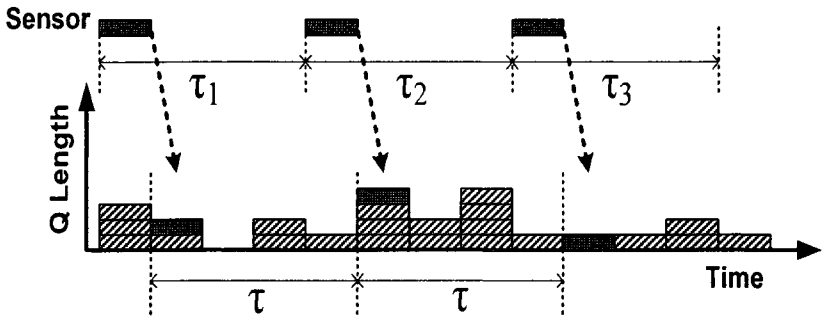


Figure 7.2 Operation of an output queue at the link shared by sensor node(s) and other cross-traffic sources

The link has an output queue, which stores packets sent into the link when the amount of arrival traffic exceeds the link capacity. The output queue has a First-In-First-Out (FIFO) link-scheduling discipline. Fig. 7.2 shows an example of the output queue in operation at the link. In the figure, the solid rectangles represent packets generated by the sensors. For simplicity, we assume that a sensor generates a single packet every time interval 'τ'. We will discuss the multiple packet case in Section V. Other rectangles represent packets from cross-traffic sources. The cross-traffic is self-similar and so variable that the amount of the cross-traffic in a certain time interval can surpass the shared link capacity. In this case, the excessive traffic must wait in the output queue, and the burstiness of the cross-traffic results in the fluctuation of the queue length as illustrated in Fig. 7.2. Meanwhile, a sensor node samples a physical environment every 'τ' unit as shown in the figure. At every sampling instant, the generated sample is packetized, and is transmitted through the communication network toward the PE. Because of the FIFO discipline, packets leave in the same order in which they arrived. In the figure, the packet generated by a sensor in the time interval τ1 arrives, and finds a packet from a cross-traffic source in the queue, so it waits for transmission in the queue. After a sensor data generation period, a new packet created in the time period τ2 arrives at the queue. In the queue, three packets belonging to cross-

traffic sources have been queued, and thus after the transmission of those packets, the packet from sensor node gets transmission. As illustrated in Fig. 7.2, the sensor-generated packets suffer from queuing delay that is induced by the amount of cross-traffic that has accumulated when the packet arrives at the queue. Using the supremum distribution of queue length from the previous studies [Fr03][Man02][No94][No95], we may not be able to properly model the periodic queuing behavior of the sensor applications. In order to model such queuing features more precisely, we propose a Probabilistic Estimation of Periodic Backlog technique.

In this section, we provide the details of the backlog estimation technique. If the output queue at the link is assumed to be infinite, i.e., it never overflows, then the unbounded queue length formed by the traffic which arrives in a time interval [s, s+τ) can be expressed as

$$Q(\tau) = A(\tau) - \alpha \cdot C\tau, \quad 0 \le \alpha \le 1. \qquad (7.4)$$

which can be justified as follows. Let $A(s, s+\tau)$ be the amount of cross-traffic that arrives at the link during the time interval [s, s+τ), where time τ is the sensor data generation period. If the amount of traffic which stays in the queue at the time s is $Q(s)$, then the total amount of traffic which contributes to the formation of the queue at time $s+\tau$ is $Q(s) + A(s, s+\tau)$. For stability of the queue, it is always assumed that $m < C$. Moreover, τ has a sufficiently large value as compare to the waiting time that backlogged packets experienced, so the leftover traffic at time s may not play a significant role to determine the queue length at s+τ. The arrival process A is stationary, so it turns out that $A(s, s+\tau)$ has the same distribution as $A(0, \tau)$, and $A(\tau)=A(0, \tau)$. Therefore, the amount of cross-traffic which contributes to the formation of queue every sensor data generation period can be represented $A(\tau)$. Park et al. show that the concentrated periods of congestion and underutilization are the trademark performance characteristics of self-similar traffic [Pa97]. Fig. 7.3 depicts an example of self-similar traffic arrival pattern.

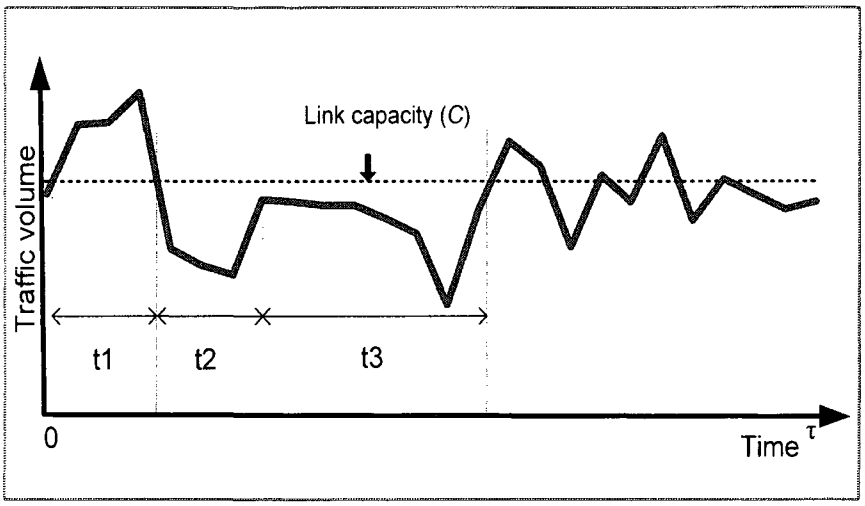


Figure 7.3 Self-similar traffic arrival

In the figure, the amount of traffic in the time period t1 is too large to be served by the link, so excessive traffic is accumulated in the queue. Because the amount of traffic arriving in the time interval t2 is small, there is sufficient space to serve the traffic queued during the time period t1. So the traffic arriving in the time interval t1, as well as new arriving traffic utilizes the link at its capacity during the time interval t2. However, in the subsequent time period, i.e. t3, the amount of traffic is less than the link capacity and there is no remaining traffic in the queue, so the link is underutilized. As seen in the figure, the link is only utilized at its capacity if the sum of the backlog in the output queue and the amount of arrival traffic at a certain time is greater than or equal to the link capacity. Otherwise, the link is underutilized. As presented in [Pa97], the presence of "peaks and valleys" at a range of time scale is detrimental to achieving high network utilization. Under the self-similar network conditions, it is difficult to maintain high link utilization. In practical cases, the amount of traffic flowing on the link in the time interval [s, s+τ) can be expressed as $\alpha \cdot C\tau$, $0 \le \alpha \le 1$. Therefore, the complementary distribution of the queue length for a queue with a service rate C fed by an FBM process with parameter *{m, a, H}* is

$$P\{Q > x\} = P\{A(\tau) - \alpha \cdot C\tau > x\}$$

$$= P\{m\tau + \sqrt{am}Z(\tau) - \alpha C\tau > x\}$$

$$= P\{Z(\tau) > \frac{x + \alpha C\tau - m\tau}{\sqrt{am}}\}. \qquad (7.5)$$

By self-similarity, we rewrite (7.5) as follow:

$$P\{Z(\tau) > \frac{x + \alpha C\tau - m\tau}{\sqrt{am}}\} = P\{Z(1) > \frac{x + \alpha C\tau - m\tau}{\sqrt{am}\tau^H}\}$$

$$= 1 - \Phi(\frac{x + \alpha C\tau - m\tau}{\sqrt{am}\tau^H}). \qquad (7.6)$$

The expression indicates that we need to know the expression of α to estimate the complementary distribution of the queue length for any FBM input traffic and an arbitrary network bandwidth. However, it is difficult to find an explicit expression for α, and hence simulation may be used in practice to determine α. In order to find a mathematical expression for α, we simulate several cases of interest and use a regression analysis on the simulation results.

In order to select the cases of interest, we consider the following. There are no significant delays in low network utilization scenarios, such as $\rho < 0.7$, and fusion application's delay targets can be met with high probability. Thus, the low utilization scenarios may be not of interest for the multi-sensor data fusion latency analysis. On the other hand, high link utilization scenarios, such as $\rho > 0.9$ are rare in today's Internet [Fr03]. Thus, we set the mean utilization levels of the links for the simulation from $\rho = 0.7$ to $\rho = 0.85$. Heyman et al. [He95] showed that long-range dependence was unimportant for buffer occupancy when the Hurst parameter was not very large ($H < 0.7$). In addition, many measurement-based studies [Fr03] show that the Hurst parameters are somewhat large, such as $H < 0.8$, especially for larger time scales. Thus, we set the range of the Hurst parameter for the simulations from H=0.8 to H=0.92. There is a wide range of for the

values of a in previous studies. However, many of the studies use around 10~20% of mean rate for the values of a [Ne98][No95]. We set a, the variance coefficient, to about 15% of the mean rate for the simulation.

Although we restrict the windows of H and network utilization ρ, it is not practical to simulate all the cases in the windows. Thus, we sample three Hurst parameter values, H=0.83, 0.86, and 0.89 from the Hurst parameter window. Similarly, we select three average utilization values from the utilization window, ρ = 0.72, 0.76, and 0.80. By simulation using the selected parameters, we find the probability, $p(x)$ that the queue size is larger than x. In order to simulate those cases, we use OMNeT++ simulator [OMNET], and generate self-similar sample paths using [Pax97]. Since the probability, $p(x)$ is the complementary distribution of the queue length, the expression (7.6) is can be written

$$1 - p(x) = \Phi(\frac{x + \alpha C\tau - m\tau}{\sqrt{am\tau^H}}). \qquad (7.7)$$

By taking the inverse function of standard normal CDF, $\Phi^{-1}(z)$ for both sides of equation (7.6), α is expressed by following equation:

$$\alpha = \frac{\Phi^{-1}(1 - p(x))\sqrt{am\tau^H} + m\tau - x}{C\tau} \qquad (7.8)$$

The queuing delay, y experienced by a packet from a sensor node is the waiting time in the queue, $y=x/C$. Fig 7.4 shows the typical shape of α when we plot it as a function of the queuing delay y. As seen in the figure, α increase as the queuing delay increase until a certain point. After the point, α seems to decrease linearly. We divide the whole region of possible values of y into two parts, and find regression functions for each region as follows:

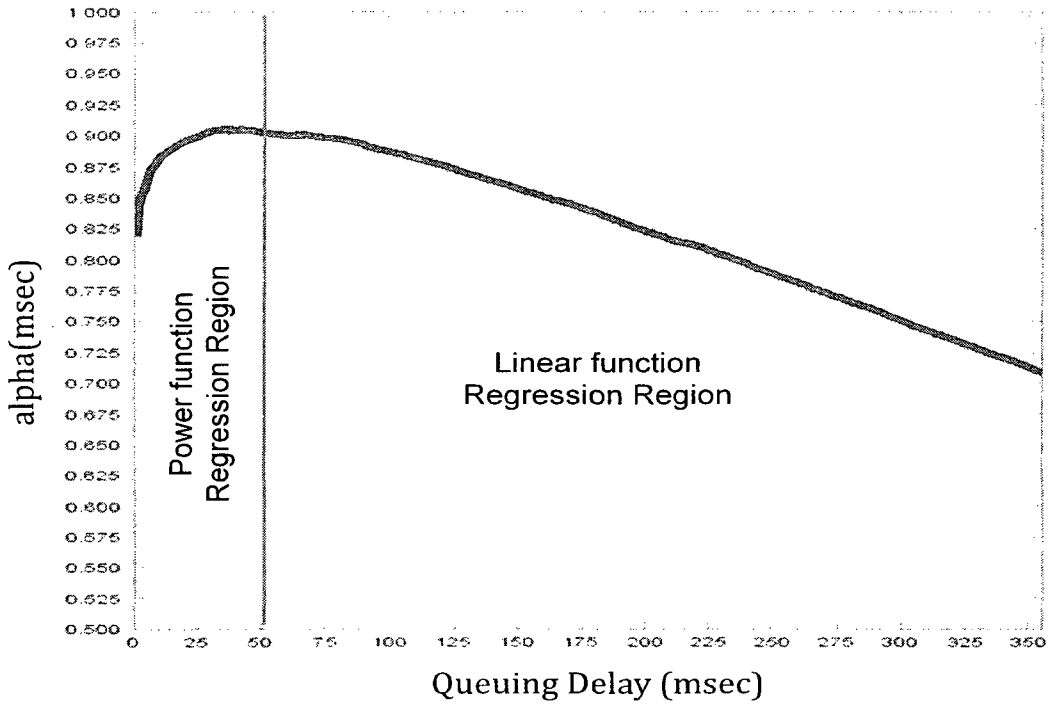


Figure 7.4 Typical shape of α as a function of queuing delay

Part 1: power function region

$$\alpha(y) = a_1 y^{b_1} + c_1 \cdot H + d_1 \cdot m + e_1 \qquad (7.9)$$

Part 2: linear function region

$$\alpha(y) = a_2 y + b_2 \cdot H + c_2 \cdot m + d_2 \qquad (7.10)$$

7.3 Single-hop delay model

The single-hop delay experienced by a packet from a sensor node is the sum of the waiting time in the queue and the transmission time for the packet into the communication link. Therefore, the distribution of the single-hop delay is found directly from the expression (7.5).

$$p(y) = P\{\frac{m\tau + \sqrt{am}Z(\tau) - \alpha C\tau}{C} + d > y\}, \qquad (7.11)$$

where d is the transmission delay for the packet and $p(y)$ is the probability that the network delay exceeds y. In order to validate the model we compare the delay computed using the model with simulation results. We consider two different sensor networks, Sensor Network 1 and Sensor Network 2. The Sensor Network 1 has a 4Mbps link and 1 second data generation period. Sensor Network 2 is connected with a 10Mbps link and a sensor node in the network takes measurements from a physical environment every 5 seconds. With the single-hop networks, we test several cases changing the input cross-traffic parameters, such as the mean rate m and the Hurst parameter H. Fig. 7.5 and Fig. 7.6 plot the delay distributions obtained from the analysis and simulation results for Sensor Network 1 and Sensor Network 2. In Fig.7.5, we change the self-similar parameter fixing the link utilization to 70%. Table 7.2 and 7.3 show the data corresponding to the results shown in Fig.7.5. As seen in the figure, the network self-similarity significantly affects the performance of sensor applications. In Fig.7.6 we change the network utilization fixing the self-similar parameter to 0.84 and 0.88 for Fig. 7.6(a) and Fig. 7.6(b), respectively. Table 7.4 and 7.5 show the data corresponding to the results shown in Fig.7.6. It is observed that the calculated results using the analytical model and the simulation results are in close agreements with each other.

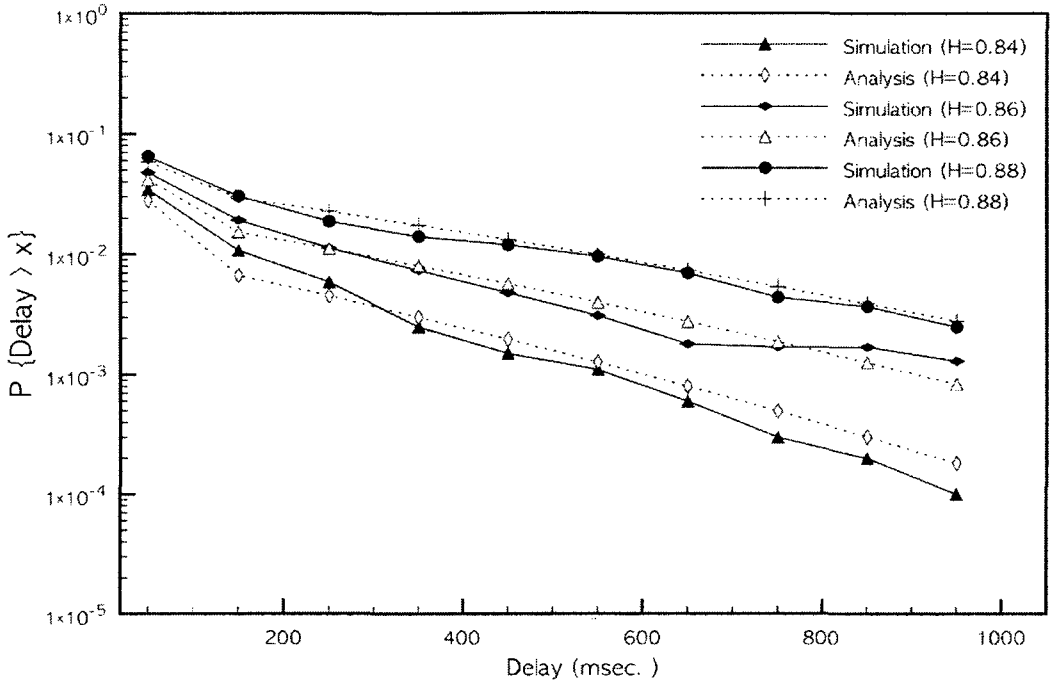
Mote-based resource limited WSN are designed to carry out one or few tasks while minimizing energy expenditure, so each sensor in a WSN generates a single packet or a small number of packets per data generation interval. However, in many of the Internet-based sensor applications, such as CASA sensors are designed for more complicated sensing tasks, and tend to generate data at high sampling rates. Thus, it is likely to generate more than one packet in a data generation period.

Delay (msec.)	H=0.84		H=0.86		H=0.88	
	Simulation	Model	Simulation	Model	Simulation	Model
50	0.0345	0.0280	0.0535	0.0454	0.0838	0.00
150	0.0108	0.0067	0.0205	0.0166	0.0371	0.0029
250	0.0059	0.0045	0.0112	0.0117	0.0231	0.0022
350	0.0025	0.0030	0.0074	0.0082	0.0153	0.0175
450	0.0015	0.0019	0.0047	0.0056	0.0119	0.0132
550	0.0011	0.0012	0.0031	0.0037	0.0087	0.0099
650	0.0006	0.0008	0.0018	0.0024	0.0064	0.0073
750	0.0003	0.0004	0.0017	0.0015	0.0040	0.0054
850	0.0002	0.0003	0.0014	0.0010	0.0028	0.0039
950	0.0001	0.0001	0.0008	0.0006	0.0023	0.0028

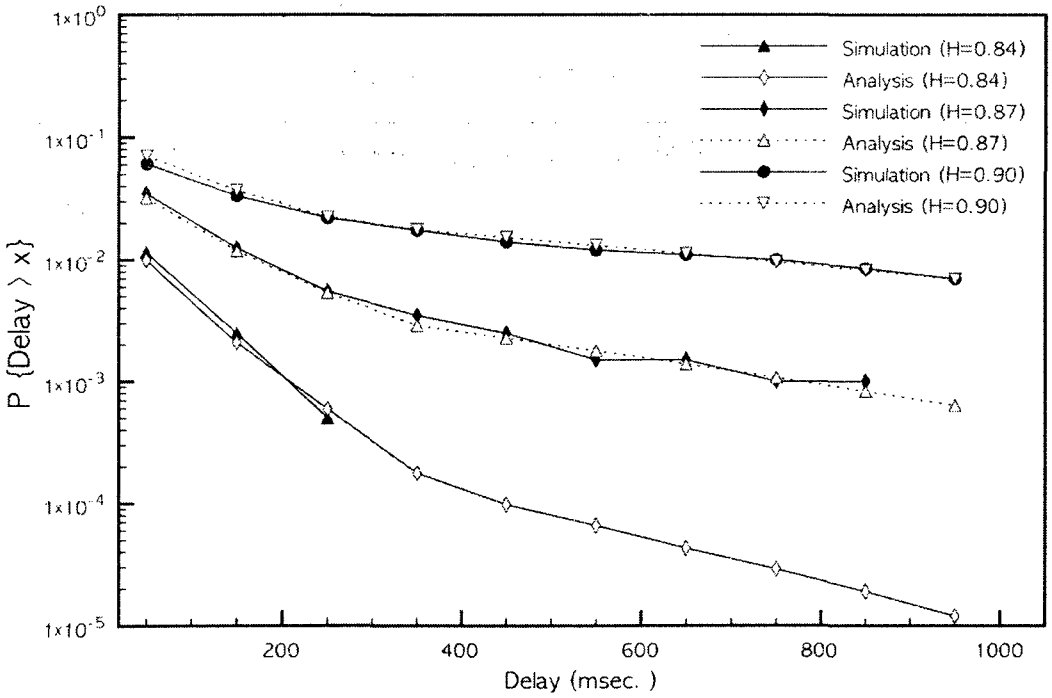
Table 7.2 Single-hop delay distributions of 4Mbps communication link with varying self-similarity parameter H.

Delay (msec.)	H=0.84		H=0.86		H=0.88	
	Simulation	Model	Simulation	Model	Simulation	Model
50	0.0115	0.0993	0.0350	0.0321	0.0610	0.0709
150	0.0025	0.0021	0.0125	0.0118	0.0335	0.0370
250	0.0005	0.0059	0.0055	0.0053	0.0220	0.0223
350		0.0017	0.0035	0.0029	0.0175	0.0176
450		0.0009	0.0025	0.0023	0.0140	0.0152
550		0.0006	0.0031	0.0018	0.0120	0.0131
650		0.0004	0.0015	0.0014	0.0110	0.0112
750		0.0003	0.0015	0.0010	0.0100	0.0096
850		0.0002	0.0010	0.0008	0.0085	0.0082
950		0.0001	0.0010	0.0006	0.0070	0.0069

Table 7.3 Single-hop delay distributions of 10Mbps communication link with varying self-similarity parameter H.



(a)



(b)

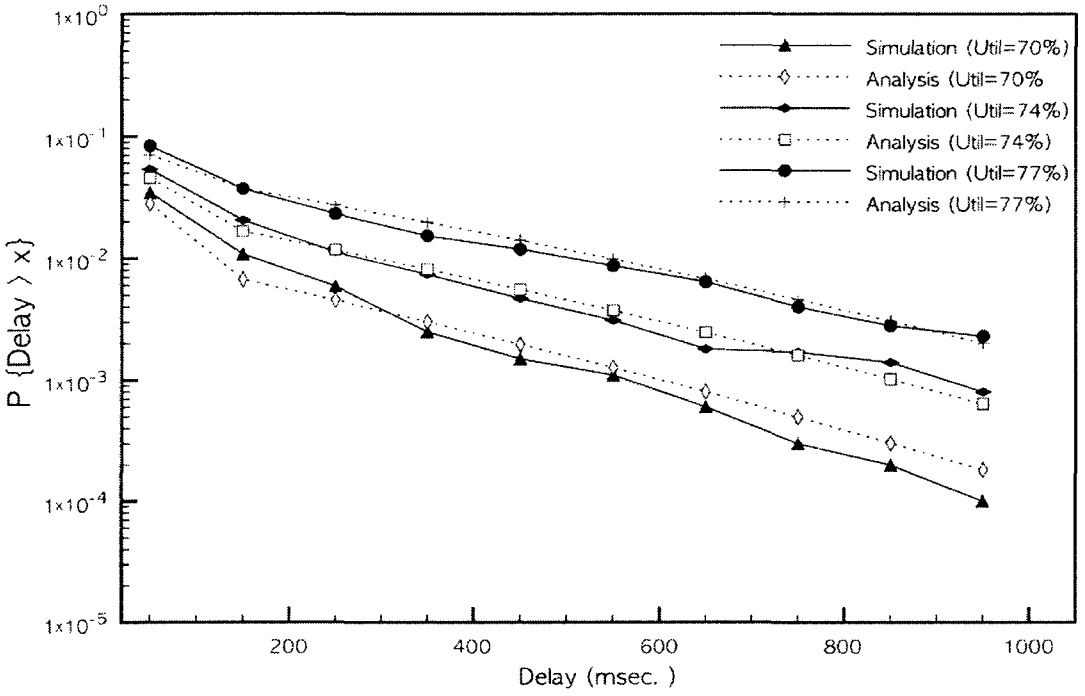
Figure 7.5 Single-hop delay distributions of 4Mbps (a) and 10Mbps (b) communication links with varying self-similarity parameter H.

Delay (msec.)	Utilization=70%		Utilization=74%		Utilization=77%	
	Simulation	Model	Simulation	Model	Simulation	Model
50	0.0345	0.0280	0.0478	0.0416	0.0650	0.0706
150	0.0108	0.0067	0.0193	0.0153	0.0305	0.0370
250	0.0059	0.0045	0.0113	0.0111	0.0188	0.0272
350	0.0025	0.0030	0.0074	0.0081	0.0141	0.0197
450	0.0015	0.0019	0.0048	0.0057	0.0120	0.0140
550	0.0011	0.0012	0.0031	0.0040	0.0096	0.0098
650	0.0006	0.0008	0.0018	0.0027	0.0070	0.0067
750	0.0003	0.0004	0.0017	0.0018	0.0044	0.0046
850	0.0002	0.0003	0.0017	0.0012	0.0037	0.0031
950	0.0001	0.0001	0.0013	0.0008	0.0025	0.0020

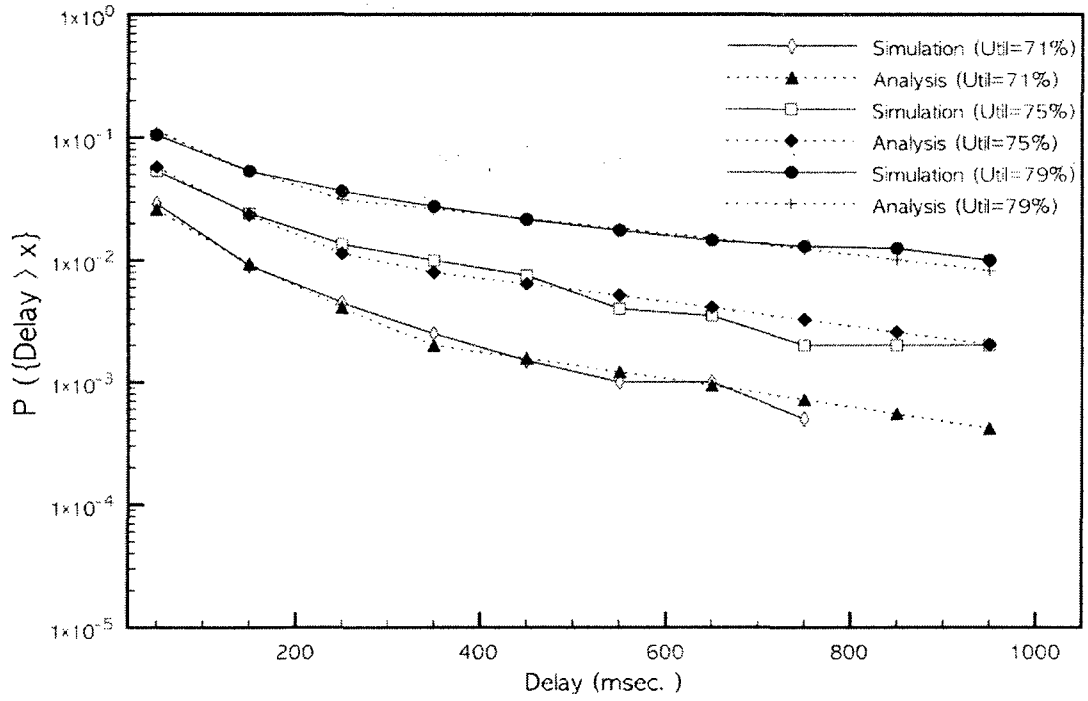
Table 7.4 Single-hop delay distributions of 4Mbps communication link with varying average link utilization.

Delay (msec.)	Utilization=71%		Utilization=75%		Utilization=79%	
	Simulation	Model	Simulation	Model	Simulation	Model
50	0.0290	0.0260	0.0535	0.0574	0.1050	0.1136
150	0.0090	0.0092	0.0240	0.0238	0.0530	0.0528
250	0.0045	0.0040	0.0135	0.0113	0.0365	0.0312
350	0.0025	0.0019	0.0100	0.0079	0.0275	0.0262
450	0.0015	0.0015	0.0075	0.0064	0.0215	0.0219
550	0.0010	0.0012	0.0040	0.0051	0.0175	0.0181
650	0.0010	0.0009	0.0035	0.0041	0.0145	0.0150
750	0.0005	0.0007	0.0020	0.0032	0.0130	0.0123
850		0.0005	0.0020	0.0025	0.0125	0.0101
950		0.0004	0.0020	0.0020	0.0100	0.0082

Table 7.5 Single-hop delay distributions of 10Mbps communication link with varying average link utilization



(a)



(b)

Figure 7.6 Single-hop delay distributions of 4Mbps (a) and 10Mbps (b) communication links with varying average link utilization

We assume that sensor data generated in a time unit consists of multiple packets and a fusion application requires all the sensor data packets before it can begin starting fusion processing. It is also assumed that each of the sensors in a network retains the same sampling rate for a while. Thus the sum of data acquisition time D and network delay N is equal to the time duration from the start of data acquisition to the arrival of the last sensor data packet at the PE. In these circumstances, as the last packet travels from the sensor to the PE along the communication link, the packet suffers from the queuing delay which is caused by cross-traffic packets and other sensor data packets which are injected into the link before the last packet transmission. The amount of traffic increases due to the sensor data, and the average traffic arrival during a sensor data generation period becomes $E[A(\tau)] = (m+s)\tau$, where s is the sensor data generation rate. If the sensor has a constant data generation rate for every data generation period, then the sensor data would not change the variance of the traffic arrival, $Var[A(\tau)]=am\tau^{2H}$, and other traffic characteristics. Therefore, the single-hop delay distribution for multiple packet cases can be expressed as follow:

$$p(y) = P\{\frac{(m+s)\tau + \sqrt{am}Z(\tau) - \alpha C\tau}{C} + d + D > y\}, \qquad (7.12)$$

where D is the data acquisition time at the sensor node. If multiple sensors share the same communication link, then the delay distribution can be expressed as follow:

$$p(y) = P\{\frac{(m+\sum_{i}^{n} s_i)\tau + \sqrt{am}Z(\tau) - \alpha C\tau}{C} + d + D > y\}$$

We will validate this approach using simulation in Section 7.4.

7.4 Two-hop Delay Model with Multiple Sources

In the previous section we developed a method to compute the delay distribution for single-hop cases. Now we address how to compute the fusion latency distribution in a two-hop network with multiple sensor nodes. We model the duration from the beginning of data acquisition at the i^{th} sensor to the arrival of the last sensor data packet at the intermediate node I using a random variable X_i. Random variables X_1, ..., X_n are independent. Similarly, we model a single-hop delay from the intermediate node I to the PE using a random variable Y. Therefore, the probability that the duration from the beginning of data acquisition to the arrival of all the sensor data at the PE exceeds a certain time z is:

$$
\begin{aligned}
&\varepsilon = P\{Z > z\} \\
&= 1 - P\{Z \le z\} \\
&= 1 - P\{\max(X_1, X_2, \ldots X_n) + Y \le z\} \\
&= 1 - \int_0^\infty P\{\max(X_1, X_2, \ldots X_n) + Y \le z \mid Y = y\} f_y(y) dy \\
&= 1 - \int_0^\infty P\{\max(X_1, X_2, \ldots X_n) \le z - Y \mid Y = y\} f_y(y) dy \\
&= 1 - \int_0^\infty P\{\max(X_1, X_2, \ldots X_n) \le z - y \mid Y = y\} f_y(y) dy \\
&= 1 - \int_0^\infty P\{\max(X_1, X_2, \ldots X_n) \le z - y\} f_y(y) dy \\
&= 1 - \int_0^\infty \prod_{k=1}^{n} P\{X_k \le z - y\} f_y(y) dy \\
&= 1 - \int_0^\infty \prod_{k=1}^{n} F_{X_k}(z - y) f_y(y) dy
\end{aligned}
\qquad (7.13)
$$

In order to validate this approach we run simulations using a network shown in Fig.7.7. The network consists of a PE, an intermediate node and four sensor nodes attached to different communication links. We consider two different sensor networks, Sensor Network 1 and Sensor Network 2 with the same network topology. The Sensor Network 1 has 4Mbps links and 1 second data generation period.

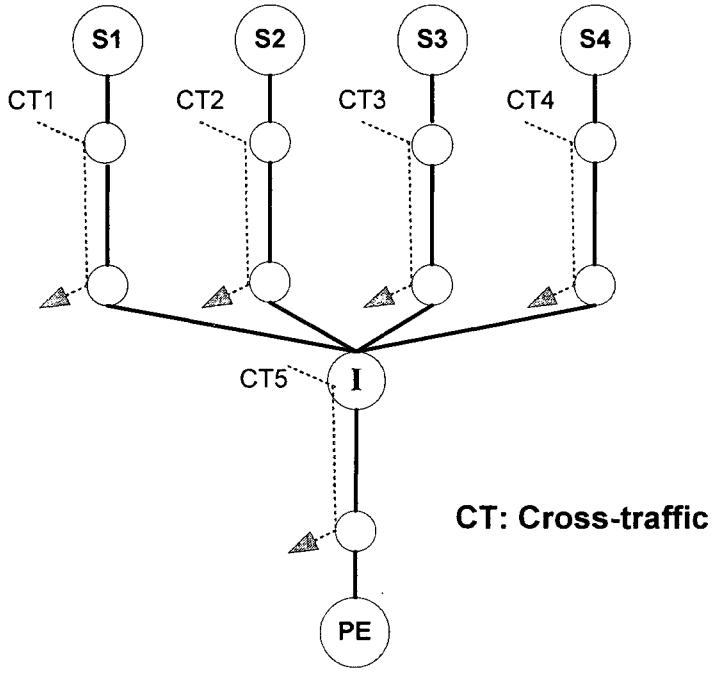


Figure 7.7 Experimental network topology

	Sensor Network 1	Sensor Network 2
S1	s=10Kbps, D=10msec	s=300Kbps, D=3sec
S2	s=5Kbps, D=5msec	s=200Kbps, D=2sec
S3	s=5Kbps, D=10msec	s=250Kbps, D=2.5sec
S4	s=10Kbps, D=5msec	s=250Kbps, D=2.5sec
CT1	H=0.80, m=2.60Mbps	H=0.87, m=7.30Mbps
CT2	H=0.85, m=2.88Mbps	H=0.82, m=7.00Mbps
CT3	H=0.86, m=2.84Mbps	H=0.85, m=7.50Mbps
CT4	H=0.89, m=2.88Mbps	H=0.90, m=7.80Mbps
CT5	H=0.87, m=2.80Mbps	H=0.86, m=7.20Mbps

Table 7.6 Parameters used for simulation

Delay (msec.)	S1 + S2		S1 + S2 + S3		S1 + S2 + S3 + S4	
	Simulation	Model	Simulation	Model	Simulation	Model
50	0.0945	0.0889	0.1188	0.1518	0.1558	0.2281
150	0.0316	0.0220	0.0408	0.0445	0.0707	0.0908
250	0.0182	0.0156	0.0233	0.0324	0.0482	0.0695
350	0.0120	0.0111	0.0159	0.0235	0.0371	0.0528
450	0.0088	0.0079	0.0114	0.0169	0.0305	0.0398
550	0.0069	0.0057	0.0088	0.0122	0.0245	0.0298
650	0.0053	0.0042	0.0073	0.0088	0.0204	0.0222
750	0.0032	0.0032	0.0060	0.0064	0.0177	0.0165
850	0.0025	0.0026	0.0044	0.0048	0.0164	0.0123
950	0.0021	0.0022	0.0032	0.0037	0.0154	0.0092

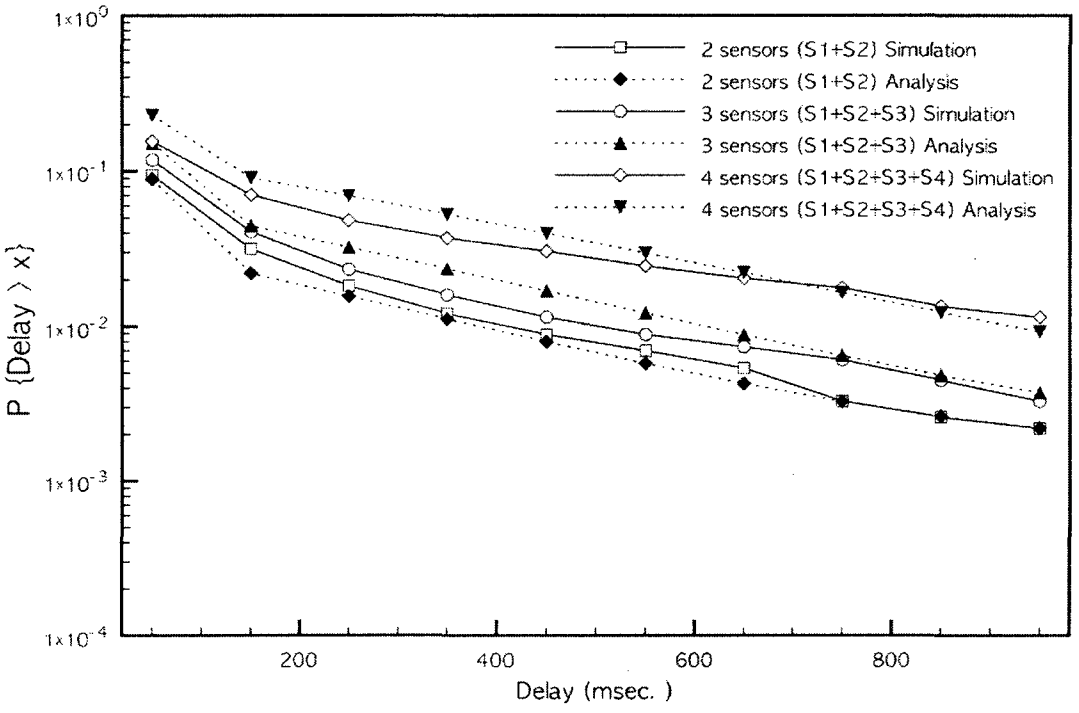
Table 7.7 Complementary delay distribution for 4Mbps network multi-sensor data fusion

Sensor Network 2 is connected with 10Mbps links and the sensor nodes in the network sample a physical environment every 5 seconds. The communication links in the network have different cross-traffic properties in terms of burstiness, i.e., self-similarity, and network utilization. In addition, the sensors are assigned to different sensing tasks, so they can have different data acquisition time and the amount of data in a data generation period. Each sensor node in Sensor Network 1 generates 5~10 packets per data generation period. While the sensor nodes in the Sensor Network 2 have higher data generation rate, and each of the sensor nodes emits 1000~1500 packets per data generation period. We set the packet length for both networks to 125bytes. Table 2 shows the properties of cross-traffic in the communication links and sensor data generation parameters. Fig. 7.8 plots the complementary fusion latency distributions obtained from the analysis and simulation results for Sensor Network 1 and Sensor Network 2. Table 7.7 and 7.8 show the data corresponding to the results shown in Fig.7.8. Comparing the analytical results with simulation results in the two cases, we can see that both results are in a close match. S4 of Sensor Network 2 is connected to a communication link which is highly

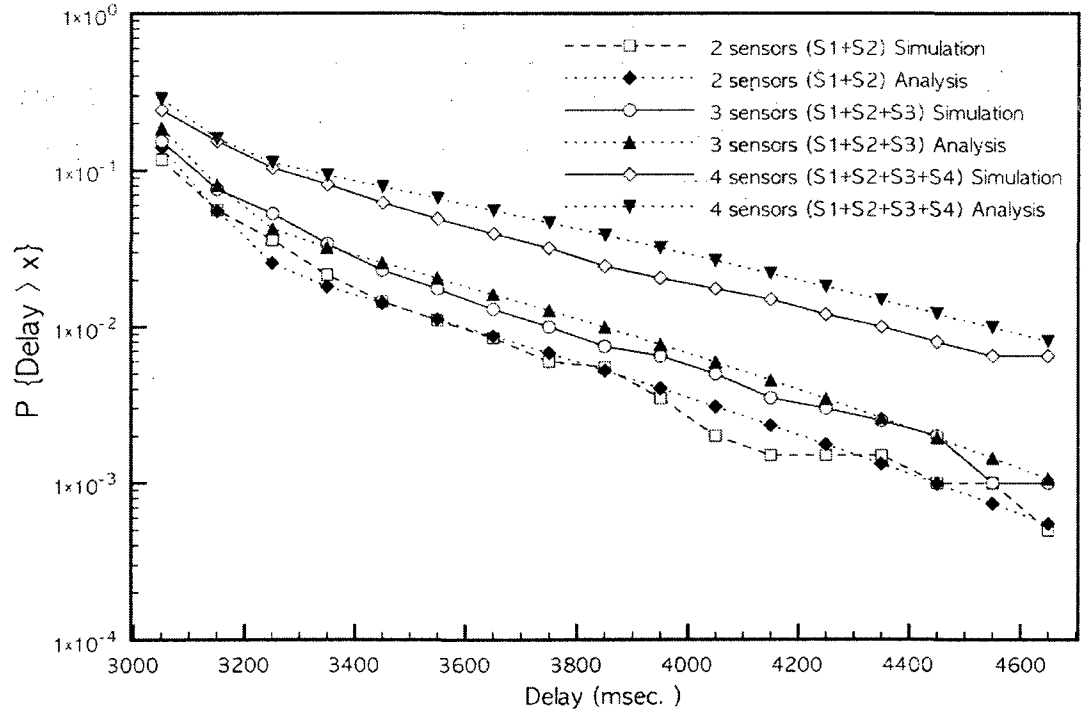
utilized and has high bursty traffic. As seen in Fig. 7.8(b), data fusions delay increases as we add the node to the network. The sensor data generation rate is 250Kbps, so the sensor node increase traffic in the link between the intermediate node I and the PE. However, only 2.5% of the link capacity is utilized by the sensor, so the traffic increment does not significantly affect other sensors' traffic. Thus, the increasing fusion latency is mainly due to synchronization delay at the PE.

Delay (msec.)	S1 + S2		S1 + S2 + S3		S1 + S2 + S3 + S4	
	Simulation	Model	Simulation	Model	Simulation	Model
3050	0.1174	0.1397	0.1539	0.1859	0.2433	0.2872
3150	0.0559	0.0551	0.0759	0.0804	0.1539	0.1597
3250	0.0359	0.0256	0.0529	0.0423	0.1039	0.1121
3350	0.0214	0.0181	0.0339	0.0322	0.0814	0.0931
3450	0.0144	0.0142	0.0229	0.0256	0.0619	0.0785
3550	0.0109	0.0111	0.0174	0.0204	0.0489	0.0661
3650	0.0084	0.0087	0.0129	0.0161	0.0394	0.0555
3750	0.0059	0.0067	0.0099	0.0127	0.0319	0.0464
3850	0.0054	0.0052	0.0074	0.0099	0.0244	0.0386
3950	0.0034	0.0040	0.0064	0.0077	0.0204	0.0321
4050	0.0019	0.0030	0.0049	0.0059	0.0174	0.0265
4150	0.0014	0.0023	0.0034	0.0045	0.0149	0.0219
4250	0.0014	0.0017	0.0029	0.0034	0.0119	0.0180
4350	0.0014	0.0013	0.0024	0.0026	0.0099	0.0148
4450	0.0009	0.0009	0.0019	0.0019	0.0079	0.0121
4550	0.0009	0.0007	0.0009	0.0014	0.0064	0.0099
4650	0.0004	0.0005	0.0009	0.0010	0.0064	0.0080

Table 7.8 Complementary delay distribution for 10Mbps network multi-sensor data fusion



(a)



(b)

Figure 7.8 Complementary delay distributions for 4Mbps (a) and 10Mbps (b) network multi-sensor data fusion

The time required for processing multi-fusion algorithms can be constant or variable depending on applications. In the simulation and the analysis we take into account the data acquisition and the data synchronization latency, but we do not consider the fusion processing time at the PE. However, the total fusion latency, including the fusion processing delay can be estimated by simply adding a processing delay distribution or a deterministic time value to the current model.

7.5 Summary

In Internet-based sensor networks, the bursty cross-traffic has significant impact on multi-sensor data fusion latency. The analysis of data fusion latency considering self-similar cross-traffic is important as it affects the quality of sensor applications. In this chapter, we proposed an analytical model for data fusion latency in Internet-based sensor networks. We present a Probabilistic Estimation of Periodic Backlog (PEPB) technique. Using the PEPB technique, we estimated single-hop delay distribution in the sensor network. Based on the single-hop delay distribution estimation, we computed the multi-sensor data fusion latency in the sensor networks considering link utilization and burstiness in cross-traffic. The multi-sensor data fusion applications require synchronizing a set of correlated data items before fusion processing can begin. The analytical model also took the data synchronization latency into account. The analytical model was validated using simulation-based results. Such analysis can be used for the design of a sensor network infrastructure to satisfy end-to-end delay requirements for mission-critical sensor applications.

In this analysis, we assumed that the communication nodes have infinite buffers, while the buffer sizes in current networks correspond to between 250ms and 1 second of queuing delay [Fr03]. In the actual networks, some of the sensor data packets would be dropped instead of being

queued incurring large delays. Therefore, a natural extension of this study is to analyze the impact of self-similar network traffic in finite buffer systems including packet loss.

Chapter 8

CONCLUSIONS

In this dissertation, we proposed an application-aware in-network service framework in order to meet the heterogeneous QoS requirements of the mission-critical sensor applications over best-effort networks. The framework consists of a general-purpose overlay architecture, Application-aWare Overlay Network (AWON), and a programming interface to support the deployment of application-aware services on the overlay nodes. We demonstrated the effectiveness of the framework by implementing a CASA application using the architecture and API. In the CASA application, high-bandwidth radar data is distributed to multiple end users with heterogeneous QoS requirements. The API was used to implement and to deploy the application-aware in-network services in an overlay network for the application. Planetlab based experiments demonstrated the suitability of the AWON architecture and the programming interface for the deployment of application-aware services. We have seen that under resource-constrained conditions and/or network congestion, an AWON-based data dissemination application could deliver better quality data to the end-users than a data-quality-oblivious implementation while using a similar amount of bandwidth. In addition our measurement showed that the AWON architecture imposed low overhead on each packet while going through AWON nodes and did not have any significant effect on the total latency from the source to the destination.

The second contribution of this dissertation was to present a design and implementation of an architectural framework for timely and accurate processing of radar data fusion algorithms in the CASA system. The framework provided a data managing and searching mechanism and a common volume finding mechanism. In addition, the framework implemented an algorithm processing module using multiple threads to meet the execution time requirements of CASA multi-radar data fusion applications. The framework is used for the real-time implementation of a CASA multi-radar data fusion algorithm, the network-based reflectivity retrieval (NBRR) algorithm in CASA-IP1. The test results of the implementation over several storm cases showed that more than 65% of the sweeps in small storm cases and about 40% of the sweeps in severe and large storm cases were executed in sub-seconds.

This dissertation then showed P2P networks are viable for data fusion in a DCAS system. In order to aggregate computing/communication resources for the resource-demanding multi-sensor data fusion applications we presented a P2P collaboration framework. In the framework, we proposed a data synchronization mechanism and a peer selection heuristic algorithm to coordinate peers for multi-sensor data fusion applications. The data synchronization mechanism selected a set of data items for data fusion considering application-specific needs. Best Peer Selection (BPS) was presented to choose a set of peers based on their computation and communication capabilities to minimize the execution time required for processing data per integration algorithm. Simulation-based results demonstrated that BPS can deliver a significant performance improvement, even though the peers have high variability in network and computation resource availability.

Another contribution of this dissertation was to develop an analytical model for data fusion response time in Internet-based sensor networks. For analyzing the multi-sensor data fusion latency, we took into account the periodic nature of sensor data generation and the network conditions which exhibit time-scale invariant burstiness (i.e., self-similarity). We developed a Probabilistic Estimation of Periodic Backlog (PEPB) technique considering the property of sensor

application and self-similarity of cross-traffic. Using the PEPB technique, we estimated single-hop delay in the sensor network. Based on the single-hop delay estimation, we developed an analytical model for data fusion latency in a sensor network which has multi-hop communication links and multiple sensor nodes. The multi-sensor data fusion applications require synchronizing a set of correlated data before fusion processing can begin. With the consideration of link utilization, burstiness in network cross-traffic, and the latency for data synchronization, the model calculated the probability that the multi-sensor data fusion latency exceeds a certain time. A comparison of the analytical model and queuing simulation-based results show that our model provides a good estimation for the multi-sensor data fusion latency.

8.1 Future Work

There are several interesting areas of research for future in both the architectural framework and in the data fusion latency analytical model. In multi-sensor data fusion applications, it is required to collect data from multiple sensors to perform data fusion algorithms. Because geographically distributed sensors generate substantial volumes of data, it may cause network congestion close to a processing node, where data fusion processing takes place. In addition, CASA is likely to support multiple data fusion applications at the same time, and these applications have different data synchronization preferences, and require different parts of data collected in the same time interval. These facts indicate that network resources should be allocated considering the application-specific requirements of multi-sensor data fusion applications. In these scenarios, significant benefits can be obtained by allowing many-to-one congestion control mechanisms in network to meet the needs of the fusion applications. AWON architecture can be used to develop application-aware many-to-one in-network services for multi-sensor data fusion applications.

Initial deployment of CASA radar network is relatively small. In current CASA scenarios, a data fusion application integrates a set of data from less than 10 different radars. Thus, the broadcasting-based peer probing mechanism in Chapter 6 did not generate significant communication overhead. We envision that the number of radars in CASA network will grow significantly, to hundreds or thousands of nodes. In such large scale networks, a data fusion application can combine data from hundreds of sensor nodes. In these scenarios, such broadcasting-based mechanisms may not be scalable and can generate unexpected loads on the network. Thus, it is required to use an efficient method for discovery of data items or resources in order to meet the requirements of mission-critical sensor applications. Structured P2P overlay networks assign keys to data items and organize peers into a graph that maps each data key to a peer [Lua05][St03][Ro01]. This structured graph can be used for efficient discovery of data items or resources using the given key in large scale sensor networks. Therefore, it would be of interest to combine a structured P2P overlay network with BPS protocol.

In Chapter 7, we assumed that the communication nodes have infinite buffers. However, the buffers in the actual networks are finite. Thus, in high utilization and high bustiness scenarios, some of the sensor data packets would be dropped instead of being queued incurring large delays. In addition, different data fusion applications have different data synchronization preferences. For example, some radar algorithms may want high sensing accuracy, so these algorithms are willing to wait for some late arriving sensor data to obtain sufficient data for enhancing accuracy. Alternatively, other algorithms may prefer timely processing, thus it is more important to fuse limited available data than to wait for some late arriving or missing sensor data. Therefore, a natural extension of this study is to analyze the impact of self-similar network traffic taking the application-specific data synchronization preferences and packet loss in finite buffer systems into consideration.

BIBLIOGRAPHY

[Ak04] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh, "A Comparison of Overlay Routing and Multi-homing Route Control," In Proc. of SIGCOMM 2004, Portland, Oregon, Aug. 30-Sept. 3, 2004.

[Ak02] I. Akyildiz, S. Weilian, S. Yogesh, E. Cayirci, "A Survey on Sensor Networks," IEEE Communications Magazine, vol. 40(8), Aug. 2002, pp. 102-114

[Al98] D, Alexander, W. Arbaugh, M. Hicks, P. Kakkar, A. Keromytis, C. Moore, S. Gunter, Nettles, and J. Smith, "The Switchware Active Network Architecture," IEEE Networks Special Issue on Active and Controllable Networks, vol. 12(3), Mar. 1998, pp. 29–36.

[Am67] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," In Proc. of AFIPS Conference, 1967, pp. 483-485.

[Am98] E. Amir, S. McCanne, R. Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding," In Proc. SIGCOMM, Sept. 1998.

[An00] D. G. Andersen, D. Bansal, D. Curtis, D. Seshan, H. Balakrishnan, "System Support for Bandwidth Management and Content Adaptation in Internet Applications," In Proc. of USENIX OSDI, San Diego, Oct. 2001.

[An01] D. G. Andersen, H. Balakrishnan, M. Kaashoek, and Robert Morris "Resilient Overlay Networks", In Proc. of ACM SOSP 2001, Banff, Canada, Oct. 2001, pp. 131-145.

[An04] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," In Proc. of IEEE/ACM International Workshop on Grid Computing, Pittsburgh, Nov. 8, 2004.

[Aw06] A. Awan, R. A. Ferreira, S. Jagannathan, A. Grama, "Unstructured Peer-to-Peer Networks for Sharing Processor Cycles," Parallel Computing, vol. 32(2), 2006, pp. 115-135.

[Ba99] H. Balakrishnan, H. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," In Proc. of ACM SIGCOMM, Cambridge, MA, Sept. 1999.

[Ba02] S. Bangolae, A. P. Jayasumana and V. Chandrasekar, "TCP-friendly Congestion Control Mechanism for a UDP-based High-Speed Radar Application and Characterization of its Fariness," Proc. 8th IEEE International Conference on Communication Systems (ICCS), Singapore, Nov. 2002, pp.164-168.

[Ba03] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," In Proc. of IEEE INFOCOM 2003.

[Ba05] Banka, T., Maroo, A., Jayasumana, A.P., Chandrasekar, V., Bharadawaj, N., and Chittababu, S.K., "Radar Networking: Considerations for Data transfer Protocols and Network Characteristics," In Proc. of AMS IIPS for Meteorology, Oceanography, and Hydrology, American Meteorological Society (AMS), 19.11. Jan. 2005

[Ban05] T. Banka, B. Donovan, V. Chandrasekar, A. Jayasumana, J. Kurose, "Data Transport Challenges in Emerging High-Bandwidth Real-Time Collaborative Adaptive Sensing Systems (poster), IEEE Infocom, 2005.

[Ba07] T. Banka, P. Lee, A.P. Jayasumana and J.F. Kurose, "An Architecture and a Programming Interface for Application-Aware Data Dissemination Using Overlay Networks," In Proc. of IEEE/Create-Net/ICST COMSWARE 2007, Bangalore, India, Jan. 2007.

[Bh97] S. Bhattacharjee, K. Calvert, and E. W. Zegura, "An Architecture for Active Networking," HPN '97, Apr. 1997.

[Bl98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, IETF, Dec. 1998

[Bo01] P. Bonnet, J. E. Gehrke, P. Seshadri, "Towards Sensor Database Systems," In Proc. of the Second International Conference on Mobile Data Management, Hong Kong, Jan. 2001.

[Br94] Braden, D. Clark and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," IETF RFC 1663, Jun. 1994

[Br01] V. N. Bringi, V. Chandrasekar, "Polarimetric Doppler Weather Radar: Principles and Operations," Camb. Univ. Press, Aug. 2001

[Br02] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and A. Vahdat, "Opus: an Overlay Peer Utility Service," In Proc. of OPENARCH, Jun. 2002.

[Br05] Brotzge, K., Brewster, B., Johnson, B., Philips, M. Preston, D. Westbrook and M. Zink, "CASA'S First Test Bed: Integrative Project #1," In Proc. of AMS 32nd Conf. Radar Meteor., Albuquerque, NM, 2005.

[Cal01] K. Calvert, J. Griffioen, B. Mullins, A. Sehgal, and S. Wen. "Concast: Design and Implementation of an Active Network Service," IEEE Journal on Selected Areas of Communications, vol. 19(3), Mar. 2001, pp.426-437.

[Ca02] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure," IEEE Journal on Selected Areas in Communications, vol. 20(10), Oct. 2002, pp. 1489-1499.

[Ca03] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," In Proc. of SOSP, Lake Bolton, New York, Oct. 2003.

[Car03] M. Carson, D. Santay, "NIST Net: A Linux-based Network Emulation Tool," ACM SIGCOMM Computer Communication Review, vol. 33(3), Aug. 2003, pp. 111-126.

[Ca06] K. Calvert, "Reflection on Network Architecture: an Active Networking Perspective," ACM SIGCOMM Computer Communication Review, vol. 36(2), Apr. 2006, pp. 27-30.

[Cas] "CASA: Collaborative Adaptive Sensing of Atmosphere," http://www.casa.umass.edu

[Ch00] Y. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," In Proc. of ACM SIGMETRICS, Jun. 2000.

[Ch02] Y. Cho, V. Chandrasekar, D. A. Vivanco, "Network Model for Clustered Radar Operating Application," In Proc. of IEEE Conference on Local Computer Networks, 2002.

[Ch04] Cheng, C-M, Huang, Y-S, Kung, H. T., and W, C-H, "Path Probing Relay Routing for Achieving High End-to-End Performance," In Proc. of IEEE Globecom, Nov. 2004.

[Ch05] V. Chandrasekar, Y.-G. Cho, D. Brunkow, A. P. Jayasumana, "Virtual CSU-CHILL Radar: The VCHILL," Journal OF Atmospheric and Oceanic Technology, vol. 22(7), 2005, pp. 979-987.

[Cl88] D. Clark. "The Design Philosophy of the DARPA Internet Protocols," In Proc. of ACM SIGCOMM, Aug. 1988, pp. 106–114.

[Cl90] D. Clark, D. Tennenhouse, "Architectural Consideration for a New Generation of Protocols," In Proc. of ACM SIGCOMM 1990.

[Cl05] D. Clark, B. Lehr, S. Bauer, P. Faratin, R. Sami, and J. Wroclwski, "The Growth of Internet Overlay Networks: Implications for Architecture, Industry Structure and Policy", In Proc. of Research Conference on Communication, Information and Internet Policy, Arlington, Virginia, Sept. 2005.

[CON] Condor, http://www.cs.wisc.edu/condor

[Cr97] M. E. Crovella, A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," IEEE/ACM Transactions on Networking, vol. 5(6), 1997, pp. 835-846.

[Ctb] "International Monitoring Network for CTBT," http://www.ctbto.org/

[Da00] V. Danjean, R. Namyst, R. D. Russell, "Linux Kernel Activations to Support Multithreading," In Proc. of IASTED International Conference on Applied Informatics (AI 2000), 2000.

[Do05] B. Donovan, D. McLaughlin, J. Kurose, V. Chandrasekar, "Principles and Design Considerations for Short-Range Energy Balanced Radar Networks", In Proc. of IEEE IGARSS05, Seoul, Korea, Jul. 2005 pp. 2058-2061.

[Du02] Z. Duan, Z. Zhang, and Y. T. Hou, "Bandwidth Provisioning for Service Overlay Networks," In Proc. of SPIE ITCOM Scalability and Traffic Control in IP Networks (II), 2002.

[Er94] H. Eriksson, "MBONE: The Multicast Backbone," Communications of the ACM, vol. 37(8), 1994, pp. 54–60.

[Fa98] T. Faber, "ACC: Active Congestion Control," IEEE Network, IEEE, vol. 3(12), 1998, pp. 61-65.

[Fa99] G. Fankhause, M. Dasen, N. Weiler, B. Plattner, B. Stiller, "WaveVideo - An Integrated Approach to Adaptive Wireless Video," In Proc. ACM Monet, Special Issue on Adaptive Mobile Networking and Computing, vol. 4(4), Oct. 1999.

[Fo00] N.L.S Fonseca, G.S. Mayor, C.A. V. Neto, "Statistical Multiplexing of Self-Similar Sources," In Proc. of IEEE Globecom, 2000, pp 1788-1792.

[Fr00] P. Francis, "Yoid: Extending the Internet Multicast Architecture," http://www.aciri.org/yoid/docs/index.htm.

[Fr03] C. Fraleigh, F. Tobagi, C. Diot, "Provisioning IP Backbone Networks to Support Latency Sensitive Traffic," In Proc. of IEEE INFOCOM, 2003.

[GEN] Genome@home. http://www.stanford.edu/group/pandegroup/genome/.

[Gu06] J.A. Gubner, "Probability and Random Processes for Electrical and Computer Engineers," Cambridge Univ. Press, 2006.

[He99] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," In Proc. of ACM/IEEE MobiCom, Seattle, 1999.

[He96] D. P. Heyman, T. V. Lakshman, "What are the Implications of Long-Range Dependence for VBR-Video Traffic Engineering?" IEEE/ACM Transactions on Networking, vol. 4(3), Jun. 1996, pp. 301–17.

[Ho05] B. Horling, V. Lesser, "Distribution Strategies for Collaborative and Adaptive Sensor Networks," In Proc. of IEEE KIMAS, 2005, pp. 497-504.

[Hw03] K. Hwang, J. In, N. Park, D. Eom, "A Design and Implementation of Wireless Sensor Gateway for Efficient Querying and Managing through World Wide Web," IEEE Transactions on Consumer Electronics, vol. 49(4), no. 4, Nov. 2003, pp. 1090- 1097.

[Gu05] E. G¨urses, G. B. Akar, N. Akar, "A Simple and Effective Mechanism for Stored Video Streaming with TCP transport and Server-side Adaptive Frame Discard," Computer Networks, vol. 48(4), 2005, pp 489- 501.

[In00] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks", In Proc. of ACM/IEEE MobiCom, Boston, MA, Aug. 2000.

[INT] INTEL, "Peer-to-peer-Enabled Distributed Computing," Intel White Paper, 2001.

[Ja91] D.N. Jayasimha, S.S. Iyengar, and R.L. Kashyap. "Information Integration and Synchronization in Distributed Sensor Networks," IEEE Transactions on System, Man, Cybernetics, vol. 21(21), Sept./Oct. 1991, pp. 1032–1043.

[Ja00] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, nd J. W. O'Toole, "Overcast: Reliable multicasting with an overlay network," In Proc. of OSDI, Oct. 2000.

[Ji06] Y. Ji, "Multi-Scale Internet Traffic Analysis Using Piecewise Self-Similar Processes," IEICE Transaction on Communications, vol. E89-B, Aug. 2006.

[Jo06]D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, K. Wehrle, "OCALA: An Architecture for Supporting Legacy Applications over Overlays", In Proc. of USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '06) San Jose, CA, May 2006.

[Ke00] R.Keller, S. Choi, M. Dasen, D. Decasper, G. Fankhauser, B. Plattner, "An Active Router

Architecture for Multicast Video Distribution," In Proc. of INFOCOM, 2000.

[Ki06] M. Kim, M. Sugaya, "IPTV in Korea and Japan," In Proc. of PTC06, 2006.

[Ko01] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, M. Lebofsky. "SETI@home - Massively Distributed Computing for SETI," Computing in Science & Engineering 3, pp.78–83.

[Ko03] D. Kostic, A. Rodriguez, J. Albrecht, A. Vahdat., "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," In Proc. of ACM SOSP, 2003.

[Kr97] A. Kruger, W.F. Krajewski, "Efficient Storage of Weather Radar Data," Software-Practice and Experience, vol. 27(6), Jun. 1997, pp. 623-635.

[Kr02] B. Krishnamachari, D. Estrin S. Wicker, "Modeling Data-Centric Routing in Wireless Sensor Networks," In Proc. of IEEE INFOCOM, 2002.

[Ku06] J. Kurose, E. Lyons, D. McLaughlin, D. Pepyne, B. Philips, D. Westbrook, M. Zink, "An End-User-Responsive Sensor Network Architecture for Hazardous Weather Detection, Prediction and Response," In Proc. of AINTEC, 2006.

[Le94] W. E Leland, M. S. Taqqu, W. Willinger, D. V. Wilson, "On the Self-Similar Nature of Ethernet Traffic," IEEE/ACM Transactions on Networking, vol. 2(1), 1994, pp.1-15.

[Le03] A. Legrand, L. Marchal, H. Casanova, "Scheduling Distributed Applications: the SimGrid Simulation Framework," In Proc. of CCGrid, May 2003, pp. 138-145.

[Lee06] P. Lee, T. Banka, A. Jayasumana, V. Chandrasekar, "Content-based Packet Marking for Application-Aware Processing in Overlay Networks," In Proc. of IEEE LCN, Tampa, FL, Nov. 14-16, 2006.

[Li04] Z. Li, and P. Mohapatra, "QRON: QoS-Aware Routing in Overlay Networks", IEEE Journal of Selected Areas in Communications, vol. 22(1), Jan. 2004, pp.29-40

[Li07] M. Li, T. Yan, D. Ganesan, E. Lyons, P. Shenoy, A. Venkataramani, M. Zink, "Multi-user Data Sharing in Radar Sensor Networks", In Proc. of ACM Conference on Embedded Networked Sensor Systems (Sensys 2007), Sydney, Australia, Nov 2007.

[Lim07]S. Lim, V. Chandrasekar, P. Lee, A.P. Jayasumana, "Reflectivity Retrieval in a Networked Radar Environment: Demonstration from the CASA IP-1 Radar Network," In Proc. of IEEE IGARSS07, Barcelona, Spain. Jul. 2007.

[Lim08] S. Lim, V. Chandrasekar, P. Lee, A.P. Jayasumana, "Real-Time Implementation of the Network-Based Reflectivity Retrieval for CASA," In Proc. of IEEE IGARSS08, Boston, MA, Jul. 2008.

[Liu04] Liu, Y., Zhang, H., Gong, W., and Towsley, D., "Application Level Relay for High-Bandwidth Data Transport," In Proc. of Workshop on Networks for Grid Applications, 2004.

[Lo04] V. Lo, D. Zhou, D. Zappala, Y. Liu, S. Zhao, "Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet," In Proc. of IPTPS, 2004.

[Lua05] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, Steven Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," IEEE Communications Survey and Tutorial, vol. 7, 2005, pp. 72-93.

[Ma02] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong, "TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks," In Proc. of OSDI, Dec. 9 - 11, 2002, Boston, MA, USA.

[Man02] P. Mannersalo, I. Norros "A Most Probable Path Approach to Queuing Systems with General Gaussian Input," Computer Networks, vol. 40, 2002, pp. 399–412.

[Ma04] P. Marino, F. Machado, S. Otero, F.P. Fontan, C. Enjamio, "Internet-based Sensor System for Environmental Monitoring," In Proc. of IEEE Sensors, Oct. 2004, pp. 24-27.

[Ma06]H. V. Madhyastha, A. Venkataramani, A. Krishnamurthy, and T. Anderson, "Oasis: An Overlay-aware Network Stack," In Proc. of SIGOPS Operating Systems Review, vol. 40(1), 2006, pp. 41-48.

[Mc96] S. McCanne, V. Jacobson, "Receiver-driven Layered Multicast," In Proc. of ACM SIGCOMM, Aug. 1996.

[Mc05] D.J. McLaughlin, V. Chandrasekar, K. Droegemeier, S. Frasier, J. Kurose, F. Junyent, B. Philips, S. Cruz-Pol, and J. Colom, "Distributed Collaborative Adaptive Sensing (DCAS) for Improved Detection, Understanding, and Prediction of Atmospheric Hazards" In Proc. of Int. Conf. on Interactive Information Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology, 11.3, Jan. 2005.

[Me99] B. Metzler, T. Harbaum, R. Wittmann, and M. Zitterbart, "AMnet: Heterogeneous multicast services based on active networking," In Proc. of Workshop on Open Architectures and Network Programming (OPENARCH), Mar. 1999.

[Na03] A. Nakao, L. Peterson, and A. Bavier, "A Routing Underlay for Overlay Networks," In Proc. of ACM SIGCOMM, Aug. 2003.

[Na06] A. Nakao, L. Peterson, and A. Bavier, "Scalable Routing Overlay Networks," In Proc. ACM SIGOPS, 2006.

[Ne98] A. L. Neidhardt, J. L. Wang, "The Concept of Relevant Time Scales and Its Application to Queuing Analysis of Self-similar Traffic," In Proc. of SIGMETRICS 98, 1998, pp. 222-232.

[No94] I. Norros, "A Storage Model with Self-similar Input," Queuing Systems, vol. 16, 1994 pp. 387-396.

[No95] I. Norros, "On the use of Fractional Brownian Motion in the theory of connectionless networks," IEEE IEEE Journal on Selected Areas of Communications, vol. 13(6), Aug 1995, pp. 953–962

[No97] B. D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, K.R. Walker, "Agile Application-Aware Adaptation for Mobility," In Proc. of ACM SOSP, 1997.

[OMNET] http://www.omnetpp.org/, "OMNeT++, A Discrete Event Simulation System"

[Pa95] V. Paxson and S. Floyd, "Wide-area traffic: The failure of Poisson modeling," IEEE/ACM Transactions on Networking, vol. 3(3), pp. June 1995, 226–244.

[Pa96] K. Park, G. Kim, and M. Crovella, "On the Relation between File Sizes, Transport Protocols, and Self-Similar Network Traffic," In Proc. of IEEE ICNP, Oct. 1996, pp. 171–180.

[Pa97] K. Park, G. Kim, and M. Crovella, "On the Effect of Traffic Self-Similarity on Network Performance," In Proc. of SPIE Int'l. Conf. Perf. and Control of Network Sys., 1997, pp. 296–310.

[Pax97] V. Paxson, "Fast, Approximate Synthesis of Fractional Gaussian Noise for Generating Self-similar Network Traffic," Computer Communication Review, vol. 27, Oct. 1997, pp. 5-18,

[Pa00] K. Park, and W. Willinger, "Self-similar Network Traffic: An Overview," In Self-Similar Network Traffic and Performance Evaluation, K. Park and W. Willinger, Eds.,Wiley-Interscience, New York, 2000, pp.1–38.

[Pla] Planet Lab. http://www.planet-lab.org

[Ptw] "Pacific Tsunami Warning Center," http://www.prh.noaa.gov/ptwc

[Qi01a] H. Qi, S.S. Iyengar, K. Chakrabarty, "Distributed Sensor Networks - A Review of Recent Research," Journal of the Franklin Institute, vol. 338(6), Sept. 2001, pp. 655-668.

[Qi01b] H. Qi, X. Wang, S. S. Iyengar, K. Chakrabarty, "Multi Sensor Fusion in Distributed Sensor Networks using Mobile Agents," In Proc. of Conference on Information Fusion, Montreal, Quebec, Canada, Aug. 7-10, 2001.

[Ro01] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems," In Proc. of Middleware, 2001.

[Sa84] J. Saltzer, D. Reed, and D. Clark, "End-to-end Arguments in System Design," ACM Transactions on Computer Systems (TOCS), vol. 2(4), 1984, pp. 195-206.

[Sa99] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat,G. Voelker, J. Zahorjan, "Detour: Informed Internet Routing and Transport," IEEE Micro vol. 19, 1999, pp. 50-59.

[SET] SETI@Home, http://setiathome.ssl.berkeley.edu

[Shr04] N. Shrivastava, C. Buragohain, D. Agrawal, S. Suri, "Medians and Beyond: New Aggregation Techniques for Sensor Networks," In Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys 2004), Aug. 2004.

[St03] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," IEEE/ACM Transactions on Networking, vol. 11(1), 2003, pp.17–32.

[Su04] L. Subramanian, I. Stoica, H. Balakrishnan, R. Katz, "OverQoS: An Overlay Based Architecture for Enhancing Internet QoS," In Proc. of NSDI, 2004.

[Te96] D. L. Tennenhouse and D. J. Wetherall, "Towards an Active Network Architecture," Computer Communication Review, vol. 26(2), Apr. 1996.

[Te97] D. L. Tennenhouse, J. M. Smith, W. D. Sicoskie, D. J. Wetherall, G. J. Minden, "A Survey of Active Network Research," IEEE Communications Magazine, 1997, pp. 80-86.

[To04] A. Todoroki, Y. Takeuchi, Y. Shimamura, A. Iwasaki, and T. Sugiya, "Fracture Monitoring System of Sewer Pipe with Composite Fracture Sensors Via the Internet," Structural Health Monitoring, vol. 3(1) , 2004, pp. 5-17.

[Ve07] E. Verplanke, "Optimizing Software for Multicore Processors," Dr. Dobb's Portal, May 2007.

[Wi97] W. Willinger, M. S. Taqqu, R. Sherman, D. V. Wilson, "Self-similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," IEEE/ACM Transactions on Networking, vol. 5, 1997, pp. 71-86.

[XBONE] Xbone, http://www.isi.edu/xbone.

[Ya06] J. Yao, J, Zhou, L. Bhuyan, "Computing Real Time Job in P2P Networks," In Proc. of IEEE LCN, Tampa, FL, Nov. 14-16, 2006.

[Yu02] B. Yu, Nahrstedt, K., "A Compressed-Domain Visual Information Embedding Algorithms for MPEG2 HDTV Streams," In Proc. of IEEE Conference on Multimedia and Expo (ICME) 2002

[Yu04] B. Yu, K. Nahrstedt, "Internet-based Interactive HDTV," ACM/Springer Multimedia Systems Journal, vol. 9(5), Mar. 2004, pp. 477-489.

[Zh99] Z. Zhang S. Nelakuditi, R. Aggarwal, R.P. Tsang, "Efficient Selective Frame Discard Algorithms for Stored Video Delivery Across Resource Constrained Networks," In Proc. of IEEE INFOCOM, 1999.

[Zi05] M. Zink, D. Westbrook, S. Abdallah, B. Horling, V. Lakamraju, Eric Lyons, Victoria Manfredi, Jim Kurose, and Kurt Hondl, "Meteorological Command and Control: An End-to-End Architecture for a Hazardous Weather Detection Sensor Network," In Proc. Of ACM Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services, Seattle, WA, USA, June 5, 2005, pp. 37-42.