Scalable, Self-Healing, and Real-Time Network Services for Directed Diffusion

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

Kenan L. Casey

Certificate of Approval:

Min-Te Sun
Assistant Professor
Computer Science and
Software Engineering

Alvin S. Lim, Chair
Associate Professor
Computer Science and
Software Engineering

David Umphress
Associate Professor
Computer Science and
Software Engineering

Yu Wang
Assistant Professor
Computer Science and
Software Engineering

George Flowers
Interim Dean
Graduate School

Scalable, Self-Healing, and Real-Time Network Services for Directed Diffusion

Kenan L. Casey

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
August 9, 2008

Scalable, Self-Healing, and Real-Time Network Services for Directed Diffusion

Kenan L. Casey

____
Signature of Author

____
Date of Graduation

Vita

Kenan Luke Casey, son of Jerry and Paula Casey, was born January 8, 1982, in Louisville, Kentucky. He graduated from Jeffersonville High School as salutatorian in 2000. He earned his Bachelor's degree in Computer Science and Mathematics from Freed-Hardeman University, Henderson, Tennessee in 2004. In 2007, he completed his Master's degree in Computer Science at Auburn University.

Dissertation Abstract

Scalable, Self-Healing, and Real-Time Network Services for Directed Diffusion

Kenan L. Casey

Doctor of Philosophy, August 9, 2008
(M.S., Auburn University, 2007)
(B.S., Freed-Hardeman University, 2004)

157 Typed Pages

Directed by Alvin S. Lim

Directed diffusion is a data-centric publish-subscribe routing protocol for sensor networks. We have proposed three network services which increase the capabilities of directed diffusion. Our protocols build on the inherent strengths of diffusion and lessen its weaknesses. The system architecture emphasizes efficient communication, local route repair, and real-time response. Our suite of network services significantly improves the performance of directed diffusion by addressing the fundamental challenges of sensor networks: energy-efficiency, dynamic environments, and scalability. Our design increases the efficiency of flooding, improves packet delivery rates in the presence of node failure, and decreases the number of packets that miss their deadlines. We evaluate the performance of our improved diffusion in terms of routing overhead, delivery effectiveness, and deadline achievement. Our results demonstrate the benefits of the network services in all three respects. We increase the efficiency of flooding by 48%, improve packet delivery rates in the presence of node failure by up to 28%, and decrease the number of packets that miss their deadlines by 30-60%.

Acknowledgments

I would like to express my appreciation to Dr. Alvin Lim for the guidance he has provided throughout my study at Auburn. I would also like to express my gratitude to the advisory committee members, Dr. Min-Te Sun, Dr. David Umphress, and Dr. Yu Wang.

Several fellow students have made significant contributions to this research including Raghu Neelisetti, Qing Yang, and Philip Sitton. I am thankful for their help and their friendship. Above all, I am grateful to my wife, Ashley, whose love and support have made this work possible. I thank her for her patience and encouragement during the long research process. Her companionship is my greatest joy.

Style manual or journal used Journal of Approximation Theory (together with the style known as “aums”). Bibliograpy follows van Leunen’s *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures

Chapter 1

Introduction

In 1991, Mark Weiser challenged computer scientists to consider a new model of computing, a model in which computers are interwoven into the "fabric of everyday life until they are indistinguishable from it" [1]. Weiser's vision, now called ubiquitous computing, emphasizes intuitive interaction with pervasive computer resources. Proactive computing is a branch of ubiquitous computing which focuses on autonomous and environmentally-based devices. Three fundamental goals for proactive computing have been proposed: get physical, get real, and get out [2]. The first goal is to connect computers to the physical world. Such computers are typically small sensors and actuators which collect data over a large physical environment and communicate it over a wireless network. The second goal emphasizes the need for real-time performance. Proactive computing systems should be faster than real-time so that feedback can be provided to automated control loops for future predictions. The final goal is to remove humans from the interactive loop in order to allow for faster-than-human response times. This change essentially shifts computation from human-centered to human-supervised.

Sensor networks have emerged in response to these lofty goals. Sensor networks are large-scale, wireless networks of resource-constrained sensor devices. They most clearly support the first goal of proactive computing, connecting the physical and virtual worlds though sensors. In reference to the second and third goals, sensor networks allow for rapid and automated response to environmental stimuli. They allow faster-than-human response by taking humans out of the loop. Sensor devices, called nodes, consist of a processor, environmental sensors, a wireless communication device (usually a radio), and a power source (usually a battery). Current sensor devices are the size of a quarter, but the goal is to further reduce their size and cost. The SmartDust project [3], for example, envisions devices one cubic millimeter in size and capable of being suspended in air. A wide range of applications for sensor networks has been proposed. Examples include environmental monitoring, military surveillance, and medical monitoring. Sensor networks are most noticeably differentiated from other types of networks by their large size, limited energy source, and dynamic nature. Sensor deployment is almost always ad-hoc due to the sheer number of sensors. Potential deployment methods include being launched by artillery shelling, scattered by ground vehicles, or dropped by airplanes. The small size of sensor devices puts severe limits on the amount of available energy so all operations are energy-efficient. Sensor networks are characterized by robustness in handling the inevitable failures which occur in the field. The system must continue to function after devices fail, die, or are destroyed.

While a great deal of sensor network research has been conducted, the inherent characteristics of sensor networks provide ample challenges for system researchers. Three of the most compelling challenges of sensor networks are their immense scale, their resource scarcity, and their dynamic topologies. Sensor networks aim to include tens of thousands to millions of nodes. To support networks of this size, communication mechanisms must be incredibly scalable. The hardware of sensor devices also provides significant challenges. Sensor devices have very limited processing, storage, and communication capabilities. Perhaps most critical is the limited energy budget available to sensor nodes. Since batteries may not be replaced in the field, all sensor network algorithms must conserve energy whenever possible. Consequently, networking protocols must maintain high computation and communication efficiency. Communication is particularly expensive in terms of energy (transmitting 1 bit consumes as much power as executing 800-1000 instructions [4]) so minimizing the number of transmissions (and receptions) is a paramount goal. Another challenge is the dynamic nature of sensor network topologies. Since sensor networks are typically deployed in an ad hoc fashion, the network must first configure itself and then maintain a working configuration in the presence of adverse environmental effects which may result in node failure. In the face of anticipated node failure, the network should continue to function normally by taking advantage of node redundancy.

The overall goal of our research is to mask the fundamental challenges of the sensor network from high level applications and application developers. Ideally, the application programmer should not know about the scale, energy, or dynamics of the sensor network. The lower layer protocols should transparently support any size network, adapt to changing energy levels, and perform dynamic reconfiguration in response to topology changes. Applications should not have to worry about such details. To address this issue, we propose three mechanisms which shield the application from the network level challenges. First, we present an efficient flooding scheme to increase the scalability of the network. Secondly, we address the dynamic nature of sensor networks through a route repair algorithm which reconfigures the network after node failure. Lastly, we propose a protocol for real-time communication which gives application developers control over data flow priorities so that time-critical messages can be delivered satisfactorily. Together, the three network services compose a new layer in the network protocol stack which applications can easily, even transparently, utilize to gain improved network performance. The network services reside slightly above the network layer but have access to the inner workings of the routing protocol. The low-level implementation of these services provides significant benefits to all the layers above the network level (i.e., transport and application). By taking advantage of the network services, any software running at higher layers will be capable of increased scalability, reactive self-healing, and real-time communication.

In Chapter 2, we discuss the motivations for our network services and describe relevant applications of sense and response systems. Chapter 3 gives background information about directed diffusion and an overview of research related to our three protocols. We describe the architecture of the proposed system in Chapter 4 and the design principles in Chapter 5. In Chapter 6, we give an overview of the implementation of the three protocols. The simulation performance of each protocol is described in Chapter 7. We conclude in Chapter 8 with a summary of our contributions and areas for future work.

Chapter 2

Motivations, Objectives, and Applications

In this chapter we discuss the motivations for our enhancements to directed diffusion. We also present several existing and potential applications of sense and response (S&R) sensor networks. Our protocols are particularly relevant to such systems since they require efficient and robust data collection as well as timely response to events.

2.1 Motivations

Although directed diffusion is generally well-suited to sensor networks, it has several apparent weaknesses. Its reliance on flooding incurs a significant penalty on communication and energy efficiency. Furthermore, diffusion also handles node failure rather poorly with its use of periodic global flooding. Finally, diffusion lacks any support for time-critical message delivery. We propose three network services to augment and extend directed diffusion to handle these issues. First, we propose an efficient flooding scheme to increase the energy efficiency and scalability of the network. Secondly, we address the dynamic nature of sensor networks through a route repair algorithm that reconfigures the network after node failure. Lastly, we present a protocol for real-time communication which gives application developers control over data flow priorities so that time-critical messages can be delivered satisfactorily. In the subsequent sections we give specific motivations for each of these mechanisms.

2.1.1 Efficient Flooding

Flooding is a packet delivery process that delivers a packet to every connected node in the network [5]. Typically, flooding requires each node to rebroadcast every flooded packet so that every node is guaranteed to receive the message at least once. This type of flooding has been called blind flooding [6] and simple flooding [7]. Efficient flooding algorithms attempt to reduce the number of redundant packet transmissions through the use of heuristics or information about the network topology. Such techniques increase algorithm complexity for the sake of communication efficiency.

Flooding has frequently been used in both proactive and reactive routing protocols. Proactive routing protocols often rely on flooding for route advertisement. In this case, every node in the network must know the current status of a link in order to maintain correct routing tables. Efficient flooding techniques are often utilized by link-state protocols since extensive neighbor information is gathered and propagated by such routing protocols [5]. By using topology information, a node can forward flooded packets to a reduced set of neighbors without affecting the global reception of the packet.

In reactive protocols, which are generally more appropriate for sensor networks, the principle reason for flooding is route discovery. Unlike proactive protocols, reactive protocols typically use blind flooding. Reactive protocols must flood initial route creation packets because no prior topology information is known. The path to an unknown host is found by flooding the network in search of the destination. Directed diffusion relies strongly on flooding since both interests and exploratory data are flooded during the gradient setup phase. After routes are discovered, diffusion performs route maintenance using the same two-phase flooding mechanism.

Due to its simplicity, blind flooding has often been implemented in reactive protocols [8] [9]. Although this naive approach to flooding simplifies the design, it results in inefficiency since nodes may rebroadcast packets that all their neighbors have already received. In blind flooding, a node may receive duplicate copies of the same flooded packet from multiple neighbors. The performance of blind flooding is inversely related to the average number of neighbors (neighbor degree) per node. As the neighbor degree increases, blind flooding results in greater redundant packets (network layer), greater probability for collision (MAC layer), and greater congestion of the wireless medium (physical layer) [10].

In dense networks it is not necessary for every node to forward each flooded packet. If only a subset of the nodes is chosen as relays, the flooding efficiency can be significantly improved. The underlying problem is to select a dominant set of nodes to relay flooded packets. More formally, we wish to find the minimal subset of forwarding nodes sufficient to deliver a flooded packet to every other node in the system [5]. This is typically accomplished with algorithms that use topology information or heuristics to identify nodes that should forward packets as discussed in Section 3.3.

2.1.2 Route Repair

Because of the dynamic nature of sensor networks, node and link failures are expected occurrences. When links fail, the routing protocol may attempt to repair from the breakage in one of two ways: end-to-end error recovery or local error recovery. End-to-end repair protocols initiate the recovery process by either explicitly alerting the source node of the problem with a negative acknowledgment or by implicitly informing the source with the absence of a positive acknowledgment. In the former case, the node which detects a break will take no action so that the sender will timeout while waiting for a positive acknowledgment. In the latter case, a negative acknowledgment will be sent from the intermediate node to the source node, reporting the link failure. Diffusion essentially uses the implicit approach (positive acknowledgment) in that the protocol defines no explicit error message depending instead on periodic route re-discovery to repair broken links.

The problem with end-to-end repair is the high cost of network-wide flooding. This has serious implications on the performance of a system in terms of scalability, energy consumption, and latency. Protocols which depend on global error recovery mechanisms do not scale well with network size. Moreover, since every node must forward the flooded packet, each node consumes energy repairing a route which is possibly very distant. Latency of end-to-end repair mechanisms also suffers since routes are completely rediscovered from the source to the sink. Figure 2.1 graphically illustrates the high cost of global route repair in two-phase pull directed diffusion. Notice the global flood of interests and exploratory data messages in both directions. Also notice that the repaired path is only a few hops different than the original data path.

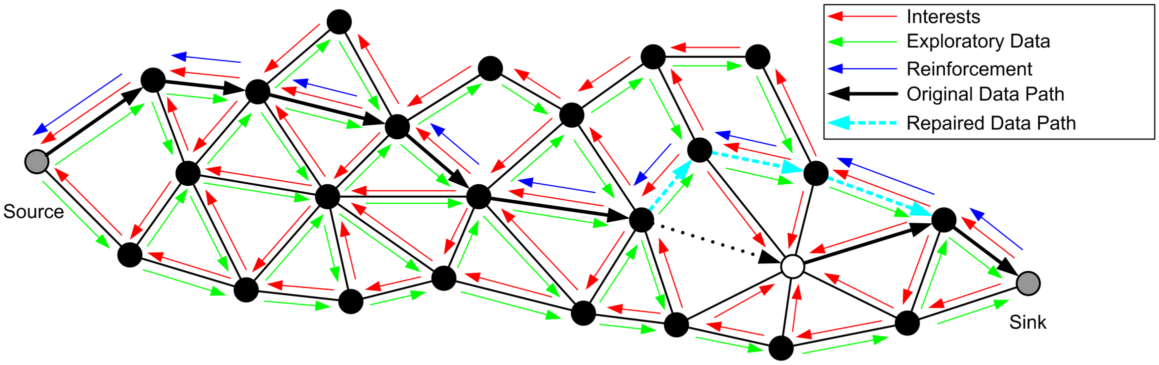

Figure 2.1: Global Route Repair

When a node relatively far away from the source fails, it makes little sense to involve the source (and other distant nodes) in the error recovery process. Ideally, only nodes around the link failure should be involved in the repair process. In cases where the repaired path is only a few hops different from the original path, such as in the previous figure, the localized approach greatly reduces the overhead associated with repair. Figure 2.2 shows an ideal case for local route repair where distant nodes are not involved in the recovery process at all. Nodes in the immediate vicinity of the break participate in the repair algorithm. Note that neither the source nor the sink are involved in the recovery process.

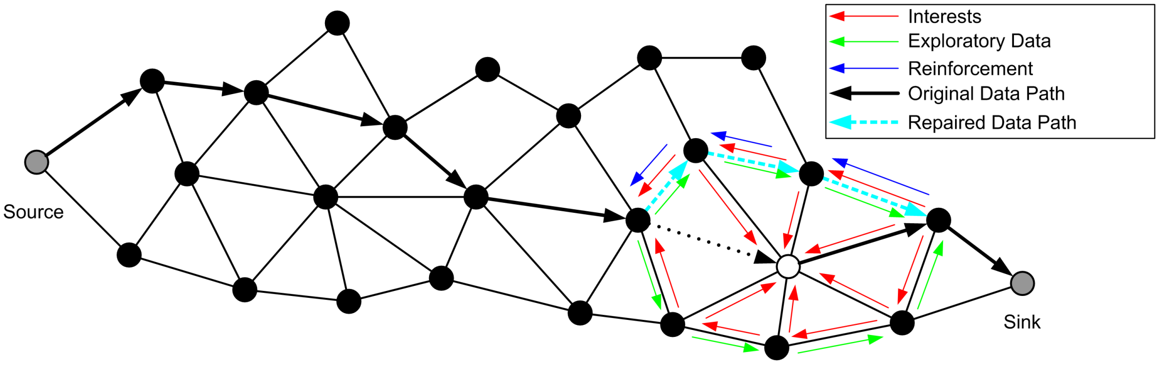

Figure 2.2: Ideal Local Route Repair

The advantages of the local repair approach include increased scalability, increased flooding efficiency, and decreased latency of repair. Since only a portion of the nodes are involved in local repair, the protocol scales more gracefully with network size and consumes less overall system energy. The localized nature of the recovery also lends itself to faster route repair since the complete (source to sink) route does not have to be traversed. In their analytical comparison of local and end-to-end recovery mechanisms, Aron and Gupta [11] show that with end-to-end recovery the probability of successfully delivering a packet on the first attempt rapidly degrades with increasing network size. In summary, local repair improves the scalability, efficiency, and latency of a network protocol. Additionally, the amount of resources consumed per packet is several orders of magnitude larger for end-to-end repair than for local repair. In summary, local repair improves the scalability, efficiency, and latency of a network protocol.

2.1.3 Real-Time Communication

The general objective of sensor networks is distributed micro-sensing, i.e. to sense, monitor, and control physical environments. Examples include acoustic surveillance systems for monitoring homes, biometric sensors which detect harmful bio-agents in airports, and stress sensors which monitor the structural integrity of buildings during earthquakes. In many applications, data messages have time constraints in the form of end-to-end deadlines specifying an upper bound on the communication delay of a packet. Application-specific deadlines, for example, allow packets associated with important events to take priority over periodic monitoring packets. Surveillance systems may require the position of an intruder to be reported to the command center within 15 seconds of detection while the bio-detection system at the airport may need to respond within 1 second. Data in monitoring systems often has an interval of validity after which it is no longer useful. The validity interval of a home surveillance system, for example, will be much shorter than that of a temperature monitoring system since the presence of an intruder will be of much greater interest immediately after his detection, but may be useless 30 minutes later.

To support this time-critical property of physical environments, sensor network protocols must implement real-time communication mechanisms. The goal of real-time communication is to minimize the number of packets which miss their end-to-end deadlines. This is typically accomplished by prioritizing packets by their deadlines so that more urgent messages are sent first. Essentially, less urgent packets are delayed so that more urgent packets can reach their destinations on time. The challenge for real-time communication over sensor networks is the multi-hop nature of sensor networks. Although it is reasonable to give priority to packets with shorter deadlines, prioritization should also be given to packets that are farther away from their destination. In order to meet its deadline, a packet with a relatively long deadline may need to be prioritized higher than a packet with a shorter deadline, if the long-deadline packet travels twice as many hops. Thus, both the deadline of a packet and the distance it must travel should be considered when packet prioritization is performed.

2.2 Objectives

In this section, we discuss the specific objectives of our three protocols. We explain the overall goal for each protocol and highlight the improvement that will be achieved by our design.

2.2.1 Efficient Flooding

Performing efficient flooding in sensor networks is challenging for several reasons. First, the communication overhead necessary to collect topology information represents a significant energy expenditure. Typically, Hello packets are exchanged among all neighboring nodes, thus incurring significant packet (and energy) overhead. Secondly, such packet exchanges occur periodically, thus incurring overhead regardless whether the network has data to transmit. The objective of our approach is to dynamically and reactively perform efficient flooding. We will use a passive clustering technique that uses ongoing traffic to create a clustered structure which allows nodes that should drop forwarded packets to identify themselves. Our approach avoids additional packet exchanges by piggybacking the clustering information on existing packets. Finally, the network performs efficient flooding reactively instead of proactively. As a reactive protocol, unnecessary transmissions are reduced. The dynamic and reactive characteristics of our approach promote energy efficiency by reducing communication.

2.2.2 Route Repair

The goal of our route repair approach is to quickly repair broken data paths in a highly localized fashion. Reactive repair is vital so that the network can quickly recover from the failure. Repair time should be kept to a minimum so that latency and delivery effectiveness are not harmed. Localized repair is essential in order to minimize the overhead associated with path repair. Thus, our route repair protocol aims to achieve localized and timely repair without impeding the scalability, efficiency, or timeliness of the network.

2.2.3 Real-Time Communication

Our goal with respect to real-time communication is to develop a real-time communication protocol which considers both distance and deadline for directed diffusion. This will allow application developers to add deadline information to a data flow and have confidence that the network will maximize the number of packets which meet their deadlines. The development of a real-time communication protocol for directed diffusion will significantly broaden the applicability of directed diffusion. Typically diffusion is limited to simple data gathering applications. This enhancement will allow diffusion networks to support complex applications which may involve timely data reporting or time-critical system response.

2.3 Applications

The emergence of pervasive computing has expanded the frontier for S&R systems. Although the S&R architecture is broadly applicable to various fields, its potential utility has not been fully realized. We present a summary of the applications of previous S&R systems and propose several new areas well-suited to the S&R paradigm.

2.3.1 Emergency Medical Response

One very compelling application of S&R systems is in the field of healthcare. The authors of [12] have developed a sensor-based emergency medical response system. At the lowest level, sensors worn by patients report vital signs and location information to local command centers (e.g. ambulances) via 802.15.4. From there, information is forwarded over a cellular or satellite link to a global command center where the data is managed as a web service. The goal of the system is to provide greater situational awareness about patient condition and arrival time so that doctors can make more informed decisions. Our real-time communication model is particularly relevant to this type of application. Since information about vital signs is being communicated, timely response of the system is critical to the care of patients. Given the large size of many hospitals, efficient flooding is also an important mechanism for satisfactory performance of S&R systems in healthcare.

2.3.2 Business Alerts

Many business applications for S&R systems have also been proposed [13] [14] [15]. [13] describes a unified event stream processing system to monitor, analyze, and detect critical business events. The objective is to improve efficiency, reduce cost, and enable the enterprise to react quickly. The Event Stream Processor calculates metrics based on event messages received from a variety of sources such as complaint databases or real-time production statistics. A domain expert creates rules based on the metrics to provide alerts for important business situations. Specific applications of this system include inventory replenishment, product behavior, and retail performance. Our networking improvements are also beneficial in the business domain. The real-time communication mechanism is well suited to the time-critical nature of many phenomena that lead to immediate response.

2.3.3 Meteorological Command and Control

S&R systems have also been developed for hazardous weather detection. Our enhanced communication protocols are especially relevant to meteorological applications where sensor failure is expected. The route repair mechanism allows the system to perform in challenging environments such as tornadoes and tsunamis. The redundant nature of the sensor network is used to overcome unavoidable node failure. Efficient flooding reduces congestion due to network flooding. This is relevant in the context of energy efficiency and network latency. Since hazardous weather detection systems will be constantly in use but infrequently activated, it is important to efficiently utilize the energy resources of the sensor devices. Efficient flooding saves energy by reducing the redundant packet transmissions. This section describes the research in the areas of tornado detection and tsunami detection.

Tornado Detection

NetRad [16] is a Distributed Adaptive Collaborative Sensing (DACS) system for early tornado detection. The goal of NetRad is to detect a tornado within 60 seconds of formation and track its centroid within a 60-second temporal region. NetRad is composed of a dense network of low-powered radars that collaborate to accurately predict tornado formation. The radars report atmospheric readings to the System Operations and Control Center (SOCC) where a merged version of the data is analyzed to detect meteorological features. Radars are re-tasked every 30 seconds based on the utility of the features identified. Thus, sensing can be focused on areas nearest to recently detected meteorological features so that a better picture of the tornado can be obtained.

The NetRad system is somewhat different from the S&R sensor network system we proposed. Unlike our S&R system, the NetRad system uses a high-bandwidth, highly-structured, wired network to connect the sensors. Our system is designed to communicate over relatively low speed, ad-hoc, wireless networks. We also emphasize fast response time on the order of subseconds, as opposed to the 30-second turnaround time of NetRad.

Tsunami Detection

Our S&R system has specifically been applied to the tsunami detection problem. In this section we describe the current tsunami warning system used by the United States and a modified system which we have proposed for better detection and response to this type of disaster.

Current System The current tsunami warning system is composed of ten buoys in the Pacific and five in the Atlantic/Caribbean. The Deep-ocean Assessment and Reporting of

Tsunamis (DART) project is maintained by the National Oceanic and Atmospheric Administration and serves as part of a tsunami warning system for the United States [17] [18] [19]. Figure 2.3 illustrates the architecture of the current system.

The DART stations consist of two parts: an anchored seafloor bottom pressure recorder called a tsunameter and a companion moored surface buoy. The tsunameter detects subtle pressure changes which indicate tsunami waves. An acoustic modem transmits data from the tsunameter to the buoy, which then relays the information via satellite to land-based warning centers. The goal of the DART system is to provide accurate and early warning for tsunamis. This includes avoiding false alarms and unnecessary evacuations.

Proposed System Our proposed system is similar to the current system, but incorporates two major modifications. First, we suggest using a multi-hop sensor network connected by radio frequency (RF) links instead of a single-hop satellite communication scheme. Although this modification necessitates a greater number of devices, it also leads to significant savings in terms of latency, energy, and cost per device. Since satellite transmission delays are avoided, the latency of communication will be improved. Shorter range radio broadcasts will result in longer battery life and greater sensor lifetime. Improved latency and energy-efficiency represent significant advantages of our design. The use of a sensor network will result in greater accuracy of detection with respect to resolution. This allows for more localized tsunami detection and prediction.

Secondly, we propose the addition of a tsunami mitigation system that responds to tsunamis. We assume it is feasible to build an artificial barrier reef strong enough to withstand the force of tsunamis and reduce its strength before it hits the shoreline. The barrier may be engaged (or fired) when a tsunami event is detected and be disengaged otherwise. Although such a barrier system would be expensive to construct, the cost may be justified for protecting particularly valuable areas (e.g. nuclear reactors). The proposed system is similar to the Thames Barrier, a series of movable gate structures which protect London from tidal floods [20], and the gate system currently being developed to shield the city of Venice from dangerously high tides [21] [22] [23].

Although the focus of our work is on the networking and analysis aspects of the system, we have also given some consideration to the design of the barrier system. We envision a defense network inspired by the concept of air bags for automobiles. The basic idea is to create a series of artificial islands which impede the progress of the wave. We propose an underwater deployment of inflation devices which are connected by wire to ballasts on the seafloor. When given the command to fire, the barriers will inflate and float to the surface while remaining tethered to the ballast. We propose using a layered series of barriers that successively attenuate the wave. Our proposed barrier system is similar to the breakwater coastal defense systems described in [24] [25]. By using advanced prediction techniques, we hope to engage the series of barriers that most effectively obstructs the wave.

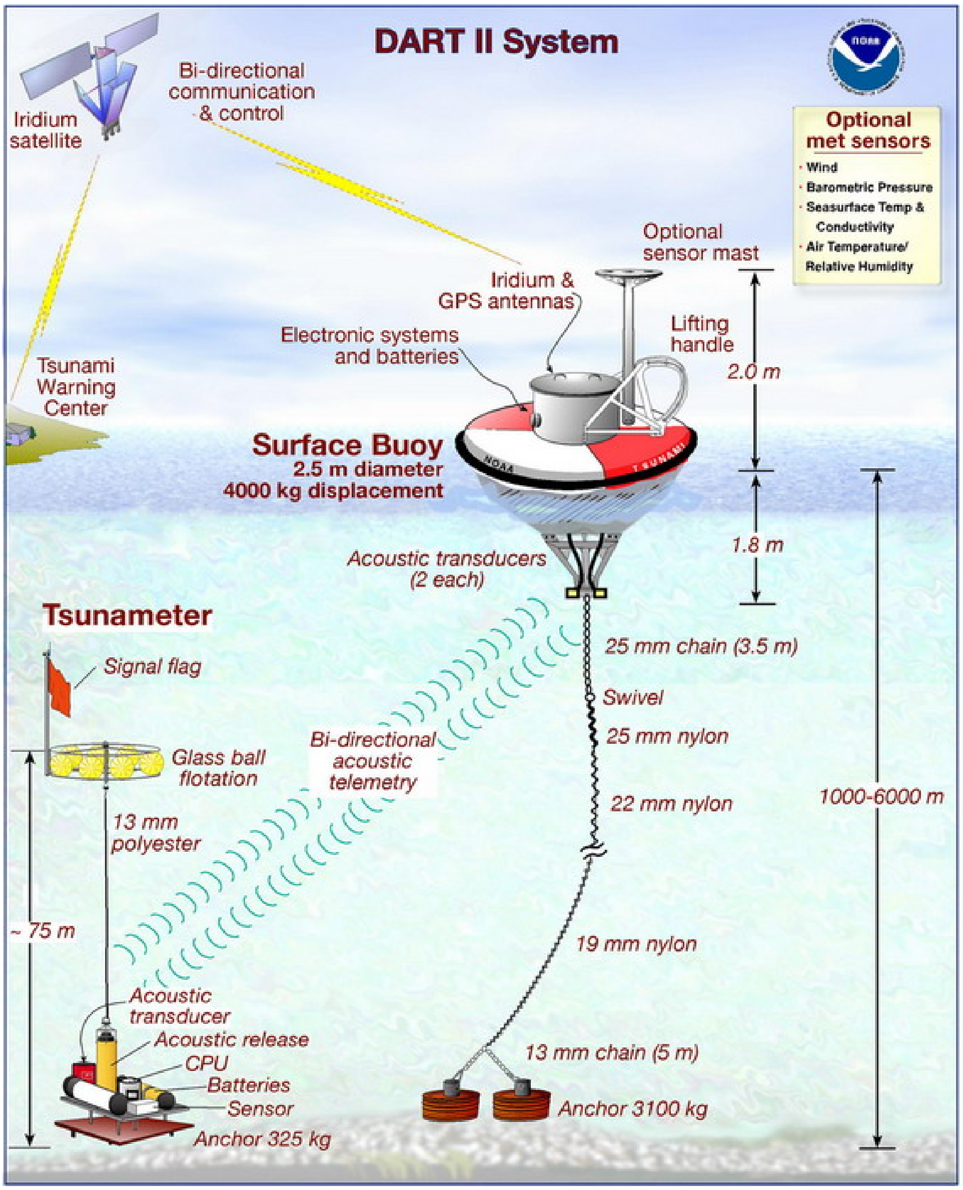

Figure 2.3: DART 2 System Architecture

Chapter 3

Related Work

3.1 Sense-and-Respond Systems

Chandy [26] presents a high-level summary of sense and respond (S&R) systems in the context of IT issues. On the most basic level, sense and respond systems simply sense the environment and respond appropriately. Rules in S&R systems can be defined using *when-then* semantics. The *when*-clause corresponds to the detection of an event, and the *then*-clause describes the execution of the response. The cost of an S&R system can be broken into three parts: the cost of false positives, the cost of false negatives, and the incremental costs of running the system. Chandy [26] outlines several important characteristics of S&R applications in order to categorize the application space. He presents the following S&R application properties:

- Response Type: Is the response human-centered or automated?
- Response Time: Is the response executed in minutes or sub-seconds?
- Event Rate: Are events generated a few per second or thousands per second?
- Condition Complexity: Are *when*-clauses based on a single event or a history of events? Are *when*-clauses based on a single stream or the fusion of multiple streams?
- Data Structure: Is the data structured in well defined schema or generally unstructured?
- Query Structure: Are queries structured or unstructured?

Our network services support S&R systems of fairly high complexity according to this taxonomy. We support an automated response time on the order of sub-seconds. Events may be generated at high rates and may compose several distinct streams of historical data that must be fused. The only exception to the trend of higher complexity is that our system utilizes structured data and structured queries.

3.2 Directed Diffusion

Directed diffusion [8] [27] is a sensor network routing protocol based on the publish-subscribe communication model. Its development was guided by the principle of data-centric routing. This is in contrast to traditional address-centric protocols which route packets based on host information. Diffusion also emphasizes in-network processing and localized interactions in order to promote energy efficiency and scalability.

3.2.1 Directed Diffusion Architecture

Protocol Overview

The objective of directed diffusion is to perform multipoint-to-multipoint communication using named data. All routing is based on named data in the form of attribute-value tuples instead of host information like address-centric protocols (e.g. IP [28], DSR [9], or AODV [29]). Diffusion is also characterized by its emphasis on localized routing. Each node stores routing information only about its immediate (i.e., 1-hop) neighbors; no global information is required.

Diffusion sets up subscriptions by flooding *interest* messages throughout the network. Nodes with matching data publish it by flooding *exploratory data* messages back through the network to the interested node. The subscribing node then reinforces its fastest neighbor by sending it a *reinforcement* message. Subsequent nodes recursively forward the reinforcement message to their fastest neighbor until the lowest latency path back to the publishing node has been reinforced. After a reinforced path has been established, *reinforced data* messages flow from publisher to subscriber via unicast or efficient multicast. After the two-phase setup algorithm is complete, data is, in essence, “pulled” from source to sink along the reinforced gradients.

Protocol Details

Diffusion uses four types of messages to establish paths within a network. A *sink* node subscribes to a data flow by flooding the network with *interest* messages that name the type of data the sink wants to receive. Intermediate nodes store the interest and record the neighbor from which it was sent. This saved path leading to the sink is known as a gradient. Nodes with data matching the interest will publish it by transmitting *exploratory data* along the gradients previously created. These publishing nodes are known as *source* nodes. When the exploratory data arrives at the sink, the sink will reinforce its single fastest neighbor (i.e., the neighbor that delivered the first exploratory data message) by sending it a *reinforcement* message. Any node receiving a reinforcement message will, in turn, reinforce its fastest upstream neighbor until a reinforced path all the way back to the source is established. Slower paths may be negatively reinforced to remove them from the gradient tables. Subsequent data emanating from the source, known as *reinforced data*, will be unicast or multicast over the reinforced path to the sink. This two-phase process results in the creation of a multipoint-to-multipoint distribution tree.

3.2.2 Strengths of Diffusion

Data-Centricity

Directed diffusion has generally proven to be well-suited to sensor networks. Its data-centric model is more appropriate for many sensor network applications. It performs more efficiently and provides more useful services for many types of sensor network applications (e.g., query processing). The use of named data provides an energy efficient mechanism for routing data, which avoids the unnecessary complexity of host information. Since data is the primary concern it makes sense to use it as the routing criterion instead of host address, a property largely unimportant in sensor networks.

Reactive Nature

The reactive nature of diffusion also supports the goal of energy efficiency. Directed diffusion belongs to a class of protocols that creates routes in response to the application's needs instead of proactively finding routes in advance. The reactive property is particularly relevant to senor networks since they may remain inactive for long periods of time waiting for events of interest.

Localized Interactions

One of the most notable characteristics of diffusion is its fully localized nature. Nodes only require knowledge about their 1-hop neighbors to create gradients. No global information, whether end-to-end or multi-hop, is needed for routing. This means that the routing table scales with the number of neighbors as opposed to the total number of nodes in the network. Node density, not network size, is the determining factor in the gradient table size. This improves the scalability of the protocol with respect to space complexity. The localized nature of diffusion also simplifies the routing protocol since global information is not required to make routing decisions.

Aggregation

Another strength of the diffusion protocol is its support for in-network processing or aggregation. Since each node understands the attribute-value tuples that compose a data message, any intermediate node may perform data aggregation. While this essentially pushes application level data down to the network layer, the resulting benefits are significant. Aggregation essentially trades communication overhead for latency by consolidating the data from multiple packets into one packet. It has been demonstrated that significant energy savings (up to 42% in [27]) can be achieved by the use of aggregation in diffusion.

Multipoint-to-Multipoint Links

A final strong point for diffusion is its support for various types of communication paradigms. Unlike other network protocols which only provide unicast (1→1) capabilities, diffusion inherently supports multicast (1→N) communication. Additionally, diffusion's publish-subscribe model can create gather distribution trees (N→1) and multipoint-to-multipoint (N→M) dissemination paths. Other protocols may support these modes of communication with high-level protocols, but diffusion's relatively simple architecture inherently supports all four classes of links.

3.2.3 Weaknesses of Diffusion

Flooding

Perhaps the greatest weakness of directed diffusion is its reliance on flooding for path creation and maintenance. In the case of standard diffusion (the two-phase pull model), both interest and exploratory data messages are flooded throughout the network whenever gradients are established. This results in a huge amount of network traffic every time a new path is established or an old path is refreshed.

Scalability

The cost of flooding rises with the size of the network, so scalability is significantly affected. Diffusion does not effectively support networks with multiple hundreds of nodes because of the huge overhead imposed by flooding across so many nodes. Although its localized nature supports scalability, diffusion's dependence on flooding severely limits the practical size of a network.

Latency of Global Repair

To deal with broken paths, diffusion periodically re-creates routes by performing the same procedure used to initially find routes: global flooding. This mechanism is described in Section 3.2.1. Since diffusion handles failure and mobility with global repair mechanisms, the costs incurred for repair are significant. To reduce the energy costs of global repair, the path maintenance mechanism is performed on a relatively infrequent schedule (e.g. every 60 seconds). In the worst case, data may be delayed an entire refresh interval before a broken path is repaired. While this may be adequate for some applications, it may be completely unacceptable for others, e.g., time-critical and real-time systems.

Lack of Real-Time Communication Support

Many applications have time deadlines which should be recognized by the network and handled differently. Directed diffusion routes packets in a first come, first serve fashion – no preference is given to higher priority flows. While not an inherent weakness, diffusion's lack of support for real-time communication is a deficiency nonetheless.

3.2.4 Route Repair Mechanisms

Standard directed diffusion includes two mechanisms for path repair. Diffusion was designed to handle path failure primarily by the periodic re-creation of gradients using a global mechanism. The designers of diffusion also mention a local repair procedure, but fail to adequately deal with the route repair problem. This section summarizes the costly global repair mechanism and the primitive local repair algorithm proposed by the original designers of the diffusion protocol.

Global Gradient Repair

The standard and currently implemented method of handling broken links in diffusion is global gradient repair. In this method, the sink periodically refreshes the path to the source by flooding the network with interest messages. The source also periodically floods exploratory data back to the sink in order to reinforce the path that is currently fastest.

Although this mechanism provides the most optimal paths, it comes at a high cost in terms of energy and latency. Global repair results in network-wide flooding of interest and exploratory data messages. To lessen this cost, global repair is performed at a fairly infrequent interval. As a result, links may remain broken for an entire gradient refresh interval. This means that a data flow may fail to report an event for 60 seconds (by default) in the worst case since no effort is made to repair the link until the next gradient refresh.

Local Gradient Repair

In [30], the designers of diffusion describe a local gradient repair strategy for the protocol. In this method, intermediate nodes may participate in reinforcement. When a node detects degradation in link quality, it can discover a new path to the source and negatively reinforce the degraded path. The problem with this scheme, as pointed out by the authors, is that every node downstream from the break will attempt to find a new path, resulting in a significant waste of resources. The authors suggest that the first nodes after the break "interpolate" data events so that downstream nodes continue to "perceive" a high quality path and do not initiate unnecessary local repair.

This method is far from ideal. It can hardly be considered local repair if intermediate nodes upstream from the break forward reinforcement messages all the way back to the source. On average, the intermediate node must reinforce half the distance between source and sink. In the worst case, the break would be 1-hop away from the sink and the so-called "local" repair would be only 1 hop different from global repair. Furthermore, without some mechanism on the part of the first node downstream from the failure, all the downstream nodes will perform local repair. The interpolation mechanism proposed to solve this problem is somewhat nebulous and largely unspecified.

3.3 Efficient Flooding

Due largely to its simplicity, blind flooding is prevalent among reactive protocols; however, because blind flooding often results in duplicate packet delivery, many resources can be conserved if a more intelligent approach to flooding is utilized. Efficient flooding techniques use topological information or heuristics to decrease the number of redundant packets transmitted during a flood. There are a variety of approaches to this problem, but we divide them into the two types proposed in [6]: heuristic-based and topology-based. Heuristic-based protocols use some sort of rule to determine whether a flooded packet should be forwarded. The topology-based algorithms make use of connectivity information to deduce which nodes need to act as packet forwarders and which nodes can drop flooded messages. We further divide the heuristic-based protocols into four subcategories and the topology-based protocols into three subcategories. In the following sections we examine each family of efficient flooding protocols and describe a few of their most important examples.

3.3.1 Heuristic-based

One approach to efficient flooding is to use heuristics to reduce the number of rebroadcasts. Heuristics based on probabilities, counters, distance, and location have been proposed [10] [31]. The main advantage of the heuristic approach is its simplicity. The primary disadvantage is the challenge of appropriately setting the parameters of the heuristic.

Probabilistic

The probabilistic scheme [10] is similar to flooding, except that nodes rebroadcast flooded packets according to some pre-determined probability p. Thus, nodes will forward flooded packets with probability p and drop flooded packets with probability $1-p$. In blind flooding, p is always 100%. In sufficiently dense networks, lower forwarding probabilities may be used without adversely affecting delivery effectiveness. In sparse networks, however, a greater probability is required for every node to receive the message.

The Fireworks protocol [32] slightly modifies the simple probabilistic scheme just described. When nodes receive a flooded packet, they broadcast it to all their neighbors with probability p and unicast it to c of their N neighbors ($c < N$) with probability $1 - p$. This modification allows for greater delivery effectiveness and finer grained control than the naive probabilistic scheme.

Counter-based

The counter-based approach [10] is another simple heuristic used to limit the forwarding of flooded packets. A counter c keeps track of the number of times a redundant broadcast message is received during some time interval. A counter threshold C is chosen as the maximum number of times a redundant message will be rebroadcast. Whenever $c \geq C$, the rebroadcast is inhibited. If the threshold has not been exceeded, the packet will be forwarded. The compelling features of this approach are its simplicity and adaptability to varying network densities. In dense areas of the network, some nodes will refrain from rebroadcasting while in sparse regions all nodes may forward the message.

Distance-based

The distance-based scheme uses relative distance between nodes as the criteria for deciding whether or not to rebroadcast. If two nodes are relatively close to each other, then the coverage area of another rebroadcast will largely overlap with the original broadcast region, and few new nodes will receive the message. However, if a node receives a message from a distant node, a rebroadcast will, for the most part, cover a different region (and therefore should reach a different set of nodes). The obvious disadvantage of this scheme is the requirement of distance estimation capabilities between each node. Ni et al. [10] claim that this may be achieved without a Global Positioning System (GPS) by the use of signal strength.

Location-based

As an improvement on the distance-based scheme, a location-based approach has also been proposed [10]. In this scheme, exact (i.e. GPS) location is used to compute a precise calculation of the additional coverage area provided by a rebroadcast. When a node sends a flooded packet, it adds its own location to the packet header. Upon reception of a flooded packet, a node calculates the additional coverage obtained by a rebroadcast and rebroadcasts if this value exceeds a coverage threshold. Otherwise, the packet is dropped. A problem with this approach is that a circle is typically used to model the communication range. Due to various environmental phenomena, this simple coverage model is rarely accurate.

Another location-based approach to efficient flooding is regional flooding [33]. In this scheme, the flooding of route discovery packets in directed diffusion is limited to a region encompassing both the source and sink. By adding the location of the source and sink to flooded messages, packets can easily be dropped when they traverse outside the region defined around the two nodes. For example, two circular regions with radii slightly greater than the half the distance between the source and sink may be defined around the source and sink as shown in Figure 3.1. This scheme has been implemented in previous research [33] as a filter using the directed diffusion filter API.

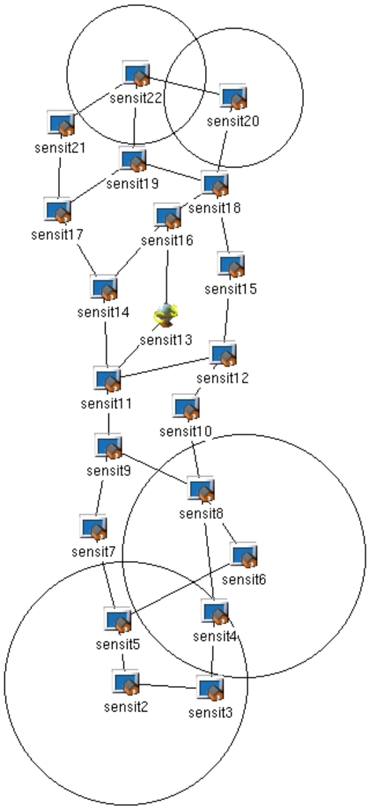

Figure 3.1: Regional Flooding (Region Filter)

Another location-based approach is the Geographic Adaptive Fidelity algorithm (GAF) [34]. The primary objective of GAF is to identify "equivalent" nodes so that some of them can be put in an energy-conserving sleep mode. From a routing perspective, nodes are equivalent when a constant routing fidelity can be maintained with only one representative node awake. Node equivalence is determined using a virtual grid which overlays the network. The grid is created with dimensions such that nodes in adjacent grids are able to communicate with each other. Since only one node per grid must be awake to maintain connectivity, nodes in the same grid are equivalent. Thus, all the nodes except one may enter a sleep state. Although not specifically an efficient flooding technique, GAF is designed to save energy throughout every phase of the routing protocol, not just route discovery.

3.3.2 Topology-based

In contrast to heuristic-based efficient flooding protocols, topology-based algorithms exploit topological information about the network to identify the best set of forwarding nodes. Most topology-based schemes use Hello message exchanges to collect topological information from neighboring nodes. Other algorithms construct a source-tree rooted at the source node and restrict leave nodes from forwarding flooded messages. A third topology-based approach involves grouping nodes into clusters in which only one member, the cluster head, is responsible for forwarding flooded packets to other cluster members.

Neighbor Knowledge-based

The neighbor knowledge-based schemes use 1- or 2-hop neighbor topology information to build a well-covered mesh of forwarding nodes. Perhaps the simplest neighbor knowledge method is Flooding with Self Pruning [35]. This protocol requires 1-hop neighbor information to be exchanged with periodic Hello packets. Each broadcast packet contains a list of the sending node's 1-hop neighbors in the header. If the receiving node cannot reach any additional nodes by a rebroadcast (i.e., its 1-hop neighbors are a subset of the set of nodes listed in the received packet), it will refrain from rebroadcasting the packet. More advanced protocols utilize 2-hop information to achieve greater flooding efficiency. The Scalable Broadcast Algorithm (SBA) [36] uses 2-hop neighbor information and the identity of the previous hop to determine if any new nodes will be reached by rebroadcasting. This may easily be determined since 2-hop connectivity is known. Dominant Pruning [35], like SBA, uses 2-hop neighbor information to make forwarding decisions. Unlike SBA, however, Dominant Pruning requires forwarding nodes to explicitly choose which of their 1-hop neighbors will be forwarding nodes. The protocol uses a Greedy Set Cover algorithm to recursively choose 1-hop neighbors which cover the most 2-hop neighbors until all the 2-hop neighbors are reached.

Multipoint Relaying (MPR) [37] is similar to Dominant Pruning in that upstream nodes explicitly notify a subset of their 1-hop neighbors to rebroadcast, but it differs in the way these forwarding nodes are selected. The only nodes allowed to rebroadcast a packet are those chosen as Multipoint Relays (MPRs). In turn, MPRs must choose a subset of their 1-hop neighbors as MPRs. The algorithm for choosing MPRs is outlined below:

1. Select all 2-hop neighbors that can only be reached by one 1-hop neighbor.

2. Select the 1-hop neighbor which covers the most 2-hop neighbors.

3. Repeat 2 until all 2-hop neighbors are covered.

MPR has been incorporated into various network protocols to improve flooding efficiency. The Open Link State Routing protocol (OLSR) [38] is a proactive link-state routing protocol designed for MANETS. By utilizing Multipoint Relays, OLSR is able to minimize the number of control messages flooded in the network and the size of messages since only links between a node and its MPR must be reported. As a link-state protocol, OLSR is capable of computing optimal routes (in terms of hop distance). Its designers claim that OLSR is appropriate for large and dense networks [38].

Simplified Multicast Routing and Forwarding (SMURF) [39] also implements MPR to help improve flooding performance. SMURF is a modular flooding component designed to complement any protocol. The MPR protocol is slighted extended to identify a connected dominating set (CDS) of nodes. A node remains in the flooding backbone (the CDS) if and only if

1. The node's ID is less than all its neighbors' IDs (or)
2. The node is the multipoint relay of its neighbor with the smallest ID.

Another protocol similar to MPR is Span. Span [40] selects a set of coordinating (forwarding) nodes that covers all 2-hop neighbors but does so in a different fashion. If a node detects insufficient neighboring coordinator nodes, it will proactively declare itself to be a coordinator to alleviate the shortage. A node's aggressiveness in becoming a coordinator is directly related to the number of neighbors it connects and its current energy level. Like GAF (Section 3.3.1), Span attempts to construct a forwarding backbone so that other (redundant) nodes can enter an energy-saving sleep mode.

Source-Tree-based

The source-tree approach involves creating a source tree with the maximal number of leaf nodes [41] [42]. The Adaptive Link-State Protocol (ALP) [41] is an example of source-tree based protocols for flooding efficiency. Unlike traditional link-state protocols, ALP does not require the state of each link to be flooded to the entire network. It uses a tree structure to disseminate link state information to only those links along paths used to reach destinations. Another source-tree-based protocol is Topology Broadcast Based on Reverse-Path Forwarding (TBRPF) [42]. TBRPF is a proactive, link-state routing protocol for mobile ad-hoc networks. A node rebroadcasts a flooded packet only if it is not a leaf node in the spanning-tree formed by the minimum-hop paths from all nodes to the source node. Constructing and maintaining the tree requires significant overhead. Nodes update their tree status with each received packet and also periodically perform blind flooding.

Cluster-based

A third topology-based approach to efficient flooding is to divide the network into a clustered structure. A representative from each group serves as a *cluster head*. Nodes belonging to two or more clusters at the same time are called *gateways*. Other nodes are called *ordinary nodes*. A cluster is defined by the transmission radius of the cluster head. Efficient flooding is achieved by restricting rebroadcasting to non-ordinary nodes (cluster heads and gateways).

The general concept of network clustering was first introduced by Ephremides et al. as the linked cluster algorithm (LCA) [43]. Cluster heads are usually elected using the Lowest ID algorithm (LID) or the Highest Degree algorithm (HD) [44]. In general, there are two approaches to clustering: active and passive.

Active In active clustering, clusters are created whether or not data transmissions are occurring. Typically, non-trivial computation and communication overhead is required to identify cluster heads and gateways. Each node must broadcast its ID (or degree) for cluster head election and compare it to the IDs (or degrees) of all of its neighbors. Periodic packet exchanges are often used to maintain the clustered structure. Communication is also necessary for gateway selection. To increase flooding efficiency, the number of potential gateways must be reduced using some gateway selection mechanism. The Flooding Gateway

Selection protocol (FGS) [45], for example, selects the best gateways using a greedy set cover algorithm and then explicitly notifies the forwarding gateways of their status.

Passive Another approach to clustering is to compose groups passively by the use of ongoing data traffic. Passive clustering (PC) is a cluster formation mechanism designed to increase flooding efficiency in mobile and ad-hoc networks [46] [5]. The protocol reactively constructs and adaptively maintains a clustered architecture to increase flooding efficiency. Unlike the active clustering protocols described previously, PC achieves flooding reduction on the fly, without explicit signaling and cluster setup packets. PC piggybacks cluster status information on data packets and constructs the cluster structure as a by-product of ongoing user traffic.

PC uses the piggybacked cluster information to deduce the role of a node as one of the three states: cluster head, gateway, or ordinary node. Cluster heads broadcast flooded packets to their neighboring nodes. Gateway nodes forward flooded packets between cluster heads. Ordinary nodes drop all flooded packets. (Their neighbors have already received the packet from a cluster head or gateway). By utilizing on-going packets to share cluster information instead of explicit control messages, PC significantly reduces communication overhead and the latency of cluster setup [5]. We describe passive clustering more thoroughly in Section 4.1.

3.4 Route Repair

In this section we summarize several protocols proposed to handle route repair in mobile ad-hoc and sensor networks. Since our emphasis is on route repair, we describe the portion of the protocol that performs path repair and omit general routing aspects of the protocol whenever possible.

3.4.1 WAR

Aron and Gupta propose the Witness Aided Routing protocol (WAR) [11] which is very similar to DSR but incorporates local correction mechanisms to recover from route failures. One of the primary goals of WAR is to avoid costly end-to-end error recovery with local repair mechanisms. WAR differs from DSR in two respects: unidirectional routing and error handling. WAR uses witness hosts to perform promiscuous route maintenance. Witness hosts are essentially routers that act on behalf of other nodes when they detect possible packet loss. For example, if host X sends to host Y and is overheard by W_1 and W_2 then W_1 and W_2 are witness hosts. If W_1 and W_2 do not hear Y forward the packet to host Z (the next hop), one of them will attempt to deliver the packet to host Z. Secondly, WAR uses a localized error recovery mechanism to repair broken paths without involving the source. When a link error occurs, the node upstream from the break broadcasts a copy of the original message with a recovery flag set. Like many other repair methods, WAR's route recovery message is constrained to a hop limited region around the repair initiator. The hop limit, denoted as the Recovery Depth, is appended to the packet. To successfully repair the link, route recovery messages must find a path to a downstream node that was on the original path. Downstream nodes can identify themselves by searching for their own ID to the route listed in the packet header. Analytical results show that as network size and route length increase, the performance of end-to-end error recovery mechanisms, degrades rapidly [11]. Hence, the local repair mechanisms of WAR support greater network scalability.

3.4.2 ADMR

An Adaptive Demand-Driven Multicast Routing (ADMR) protocol [47] is chiefly concerned with delivering packets to a multicast group in an on-demand fashion (instead of continuously maintaining the multicast group structure). ADMR routes messages through a tree structure from the source (root) to each group member (leaf or branch). When forwarding nodes or receiving members become disconnected from the multicast forwarding tree, ADMR invokes a local subtree repair algorithm to detect and repair the path. Each node maintains a *disconnection timer* for each group based on the inter-packet arrival time. If no packet is received within this time interval, ADMR assumes disconnection has occurred. Nodes that detect disconnection initiate local repair by sending a repair notification packet to nodes below them in the subtree (downstream). After sending a Repair Notification, nodes wait for *repair delay* period of time. If a Repair Notification is received within this time interval, the node will cancel its local repair (since it is further downstream from the break). No Repair Notification will be received by the node whose parent has failed so it will identify itself using this procedure and initiate local repair. This node will flood a hop-limited Reconnect packet in the neighborhood around itself. When nodes along the original path receive Reconnect packets, they forward them without incrementing the hop count. Thus, Reconnect packets can travel back to the source along the original path. If the original path is found by the hop-limited flood, the source will respond with a Reconnect Reply packet which will be unicast back to the repair node along the reverse path the reconnect message took. In this way, a path from the source (root) to the destination around the broken link will be reconstructed.

3.4.3 SWR

A Single path With Repair routing scheme (SWR) is proposed in [48]. This protocol is motivated by the desire to avoid source-initiated path repair. In SWR, the pivot node, located immediately upstream from the break, searches for alternate paths around the broken link. It does so by broadcasting a Help Request (HREQ) to its neighbors. Upon reception of an HREQ message, nodes use previously stored information about the topology of the network to create an alternate path around the failed node with Help Response (HREP) packets. SWR maintains topological knowledge of the network by the use of cost information transmitted in each data packet. This procedure is recursively repeated upstream back to the source if the original pivot node is unable to find an alternate path.

3.4.4 SHORT

A framework of Self-Healing and Optimizing Routing Techniques (SHORT) is presented in [49]. Unlike other repair mechanisms, SHORT attempts to find new routes constantly, even when connectivity is present. In some sense, it attempts to heal broken links before they break by constantly searching for better paths. SHORT can be applied on top of existing mobile ad-hoc routing protocols (e.g. DSR or AODV) to increase performance by optimizing existing routes when a better local sub-path becomes available. Two algorithms are described which optimize proactive repair based on path length (Path Aware-SHORT) or energy level (Energy Aware-SHORT). The primary goal of this approach is to discover short-cut routing paths. Short-cuts result from the mobility of nodes in the ad hoc network. Although connectivity may still be intact, the shortest path from source to destination may change after its initial discovery. SHORT continually seeks and takes advantage of such short-cuts. This is accomplished through the use of a hop count (HC) field on each packet. Nodes maintain a hop comparison array for each data flow to identify short-cuts. The addition of SHORT improved the performance of DSR and AODV in terms of both path optimality and message delivery rate [49].

3.4.5 RDMAR

Relative Distance Micro-discovery Ad Hoc Routing (RDMAR) [50] localizes flooding of route discovery queries by estimating the relative distance between the source and destination. This approach to limiting route discovery and repair floods is the distinctive characteristic of RDMAR. By assuming a maximum velocity and average transmission range of all mobile nodes, RDMAR calculates the maximum number of hops separating two nodes. RDMAR requires each node to maintain a routing table with information about every other host in the network. This includes the following fields:

- Default Router - Neighbor indicating the next hop for this destination host.
- Relative Distance - Estimate of the relative distance (in hops) to this destination host.
- Time of Last Update - Time last update was received from this destination host.
- Route Timeout - Time remaining before route is considered invalid.
- Route Active - Flag indicating whether the route is currently active.

RDMAR uses the Time of Last Update to compute a time interval of uncertainty designated as t_{motion}. Assuming a velocity $Micro\_Velocity$ and a transmission range $Micro\_Range$, the source and destination nodes can estimate their minimum and maximum radius of movement during time period t_{motion} as shown in Figure 3.2. This distance is divided by the $Micro\_Range$ to compute the number of hops that should be traversed in the route discovery flood. RDMAR utilizes the same Micro-Discovery mechanism for route repair except that it is initiated by an intermediate node instead of the source. Upon detection of a link failure, an intermediate node will flood route requests to the destination using relative distance information from its routing table to set the number of hops appropriately. In this way, route requests will be restricted to the region between the intermediate node and the destination.

3.4.6 ABR

Associativity-Based Routing (ABR) [51] [52] defines a routing metric called *degree of association stability* used to make routing decisions. Association stability is related to the connection stability of one node with respect to another node over time and space. The destination examines association stability of potential routes and chooses the one which is most stable and contains the fewest number of hops. ABR is divided into three phases: route discovery, route re-construction, and route deletion. The second phase, route re-construction, deals with route repair in a manner very similar to SWR (Section 3.4.3). The node immediately upstream from a break broadcasts local (hop-limited) queries that perform partial route discovery. This process is repeated recursively upstream toward the source. The backtracking process is discontinued if the node currently performing the repair

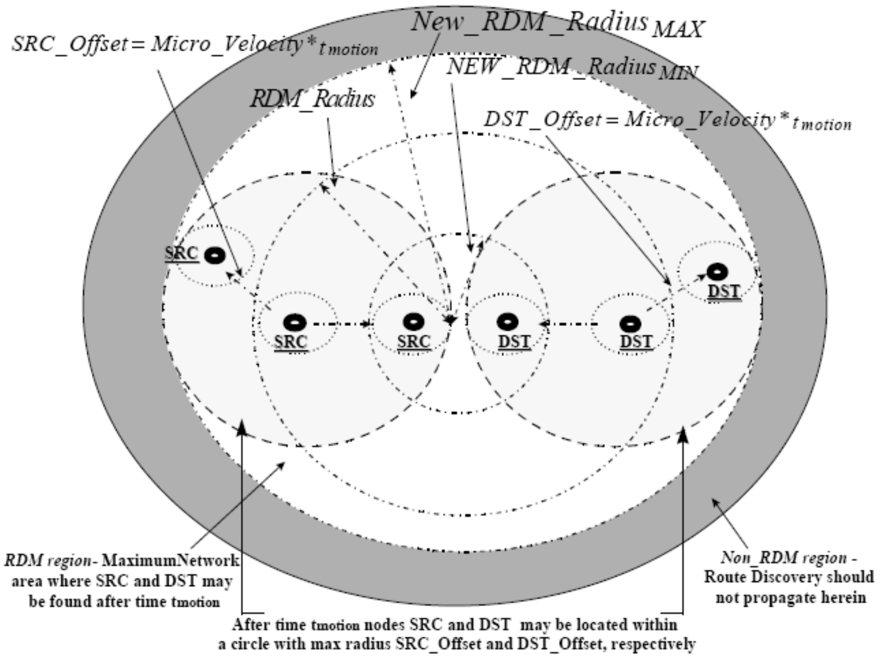
SRC_Offset = Micro_Velocity * t motion
New_RDM_Radius MAX
NEW_RDM_Radius MIN
RDM_Radius
DST_Offset = Micro_Velocity * t motion
SRC
SRC
SRC
DST
DST
DST
RDM region- MaximumNetwork area where SRC and DST may be found after time tmotion
Non_RDM region - Route Discovery should not propagate herein
After time tmotion nodes SRC and DST may be located within a circle with max radius SRC_Offset and DST_Offset, respectively

Figure 3.2: Relative Distance Micro-Discovery

is more than half the distance (in hops) from the destination to the source. In this case, global route discovery is performed instead of localized repair.

3.4.7 TORA

TORA (Temporally-Ordered Routing Algorithm) [53] is a source-initiated routing protocol based on the concept of link reversal. Routes are created using a height metric to establish a Directed Acyclic Graph (DAG) rooted at the destination. Links are assigned a direction (either upstream or downstream) based on their relative height. In order to create this structure, nodes need information about their 1-hop neighbors. TORA incorporates a route maintenance mechanism that supports localized route repair. The node immediately upstream from the link failure generates a new reference level, which is propagated by neighboring nodes and causes nodes to adjust to the new height. When a node has no downstream links, it reverses the direction of its links. This link reversal propagates upstream until a new route to the destination can be found.

3.4.8 AODV-LRQT

Pan et al. [54] present an extension to the local repair mechanism for the Ad-hoc Distance Vector (AODV) routing protocol called AODV-LRQT. The modified protocol limits the extent to which the repair mechanism is applied along two network dimensions: depth and breadth. Both approaches limit the scope and cost of flooding. Decreasing the depth means limiting the number of times a node can forward a repair route request. This is similar to the counter-based efficient flooding technique discussed in Section 3.3.1. To reduce the breadth of repair, route repair packets are given a hop count, or time to live (TTL), which localizes the scope of potential new paths to some n-hop neighborhood around the broken link. These mechanisms are implemented using two algorithms: repair quota and adaptive TTL. Each node has a repair quota (RQ) to control the breadth of repairs. When the RQ has been met, a node will no longer forward route request packets. Depth reduction is implemented with an adaptive TTL. The algorithm assumes knowledge of the network topology and transmission range in order to find a hop count which is half the length of the longest path in the network. Simulation showed that, in AODV, constraining the breadth (repair quota) of route request floods provided a greater performance improvement than constraining the depth (adaptive TTL) [54].

3.4.9 PATCH

Proximity Approach To Connection Healing (PATCH) [55] is another local recovery mechanism proposed for DSR. It aims to reduce the control overhead and achieve fast, localized recovery. When an intermediate node detects a broken link, it floods a *local recovery request* in the two-hop region around itself, looking for downstream nodes on the path (i.e. those closer to the destination). It does so by including in its header a list of all the nodes between the intermediate node and the destination. If one of the nodes included in the header receives the *local recovery request*, it sends a *local recovery reply* to the source so that the new route will be used in future communication. If no route is found within some recovery interval, the source will initiate the end-to-end error recovery process.

3.5 Real-Time Communication

3.5.1 RAP

RAP is a real-time communication architecture for large scale wireless sensor networks proposed in [56]. The goal of RAP is to provide a scalable and lightweight communication service that maximizes the number of packets meeting end-to-end deadlines. The network stack for the RAP protocol suite is shown in Figure 3.3.

On the top layer, a query-event service allows application developers to easily monitor events and submit queries to the sensor network. Queries are registered beforehand and triggered when an event occurs. When an event that matches the attributes of interest occurs in the geographic area of interest, a message stamped with the timing constraint is sent to the base station at the registered location. Queries are written and registered using the following API:

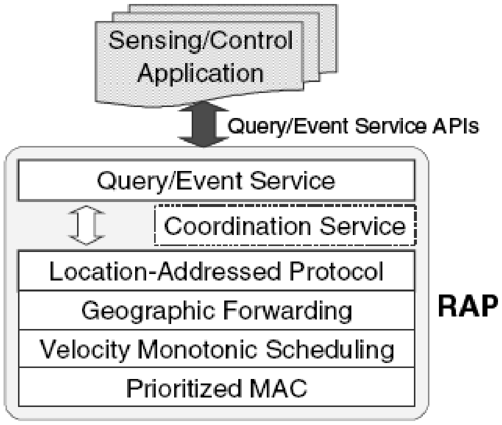

Figure 3.3: RAP Communication Architecture

- query(attributes, area, timing_constraints, base_station_location)
  - attributes - Attributes of interest.
  - area - Geographic area of interest.
  - timing_constraints - Timing constraints for event, i.e., end-to-end deadline.
  - base_station_location - Location of base station which will receive event data.
- register_event(event, area, query)
  - event - Name of event.
  - area - Geographic area of interest.
  - query - Query associated with event.

The Location-Addressed Protocol is a transport layer similar to UDP except that it uses location information instead of IP addresses for identifying hosts. Routing is handled by

Geographic Forwarding (GF), a simple but robust network protocol which forwards packets based on location information. GF greedily forwards packets to the neighbor closest to the packet's destination. GPSR is used to route packets around the perimeter of a void region.

The heart of RAP is Velocity Monotonic Scheduling (VMS), the layer that provides support for real-time communication. VMS is the packet scheduling policy that determines the order in which incoming packets are forwarded. Typically, ad hoc networks forward packets in FCFS order but this policy performs poorly in networks where data flows have different end-to-end deadlines. In contrast, RAP prioritizes packets based on their "local urgency." VMS considers both the temporal deadline and the geographic distance when scheduling packets. Thus, VMS is both deadline-aware and distance-aware. This means that packets with shorter deadlines and packets with longer distances to the destination will have higher priorities. VMS defines the velocity of a packet as the quotient of the distance to the destination and the time deadline. By assigning priorities based on the velocity, VMS is able to accurately quantify the "urgency" of a packet and thereby meet more deadlines. Two priority assignment policies are defined in VMS: static velocity monotonic (SVM) and dynamic velocity monotonic (DVM). SVM calculates a fixed velocity once at the source before the packet is transmitted. SVM computes the velocity using Equation 3.1 where $p_{src} = (x_{src}, y_{src})$ and $p_{dst} = (x_{dst}, y_{dst})$ are the locations of the source and destination respectively, $\|\cdot\|$ is the distance between two points, and $T_{deadline}$ is the end-to-end deadline.

$$V = \frac{\|p_{src} - p_{dst}\|}{T_{deadline}} \tag{3.1}$$

DVM re-computes the velocity at each intermediate hop using Equation 3.2. Note that T_i represents the time elapsed for the packet to reach intermediate hop i. Initially,

$T_i = T_0 = 0$ and $p_i = p_{src}$ at the source node. By updating the priority at each node, a packet that is progressing more slowly than its velocity may dynamically increase its priority. Likewise, a packet traveling more quickly than its requested velocity may be slowed down to give way to more urgent packets.

$$V = \frac{\|p_i - p_{dst}\|}{T_{deadline} - T_i} \tag{3.2}$$

To implement VMS, the network must use a prioritized queue. RAP prioritizes at two levels to maximize performance. Prioritization based on the velocity is performed at the network layer and at the MAC layer. The network layer places packets in a queue ordered by velocity (higher velocity first). Additionally, RAP extends the IEEE 802.11 MAC protocol to adapt the wait time (DIFS) and backoff window (CW) based on the priority of the packet.

RAP has been shown to significantly reduce the deadline miss ratio when compared to traditional FCFS mechanisms. When compared to DSR over standard IEEE 802.11, RAP reduced deadline miss ratio from 90.0% to 17.9% [56]. Despite its simplicity, RAP significantly increases the real-time performance.

3.5.2 SPEED

SPEED [57] is another real-time protocol for sensor networks. SPEED employs feedback control and stateless algorithms to support soft real-time communication. The protocol's design also emphasizes load balancing, localized behavior, and minimized dependence on the MAC layer. Like RAP, SPEED uses a location-based routing protocol to forward packets. The organization of the SPEED protocol suite is illustrated in Figure 3.4.

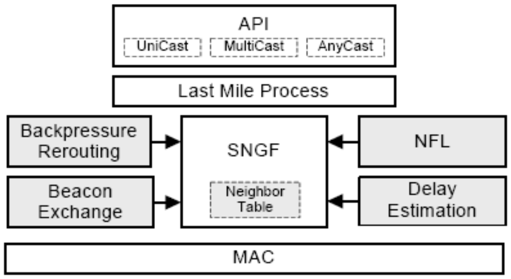

Figure 3.4: SPEED Architecture

At the top layer, SPEED provides an API that supports three types of communication modes: unicast, area-multicast, and area-anycast. While unicast follows the familiar 1-1 communication model, multicast and anycast are somewhat unconventional modes based on geographic constraints. Area-multicast delivers a packet to each node in a circular region defined by a center position and radius. Area-anycast is designed for applications in which it is sufficient for one node in some area to respond for that region. Like other geographic routing protocols, every node in SPEED participates in a periodic beacon exchange with its neighbors to share location information. Two on-demand beacons, a delay estimate beacon and a backpressure beacon, are also employed by the protocol. By timestamping each data packet and ACK, the single hop delay to each neighbor can be calculated. Instead of using queue size to gauge congestion, SPEED uses this single hop delay as a metric to approximate load in the network.

Routing in SPEED is handled by Stateless Non-deterministic Geographic Forwarding (SNGF), a modified version of simple Geographic Forwarding (GF). SNGF defines the forwarding candidate set, FS_i, of node i as the set of neighbors of node i that are closer to the destination. Relay Speed, *Speed*, is computed by dividing the gain in distance to node

j by the estimated time delay to node j. Formally,

$$Speed_i^j = \frac{\|\vec{p_{dest}} - \vec{p_i}\| - \|\vec{p_{dest}} - \vec{p_j}\|}{HopDelay_i^j} \tag{3.3}$$

where p_{dest} is the location of the destination, p_i is the location of node i, $\|\cdot\|$ is the distance between two points, and $HopDelay_i^j$ is the estimated delay from node i to node j. Packets are forwarded to nodes in FS_i based on their relay speed. The node with the maximum relay speed is chosen as the next hop if the relay speed is greater than some $S_{setpoint}$, a system parameter. Otherwise, the packet is probabilistically forwarded to the neighbor with the highest relay speed. If the packet is not forwarded, backpressure rerouting is initiated. This algorithm works with the neighborhood feedback loop (NFL) to adapt the network layer to congestion. When congestion is detected by NFL, backpressure beacons will be sent upstream to find routes around the congested area. SPEED uses this same mechanism to discover routes around network voids. Although SPEED is somewhat more complex than RAP, it provides several advanced real-time and congestion-avoidance features not offered by RAP. In some sense, SPEED is a heavyweight approach to real-time communication while RAP is more a lightweight protocol.

3.5.3 Other Protocols

MMSPEED [58] is an extension of SPEED which provides support for different levels of timeliness and reliability. Timeliness is achieved using the required delivery speed algorithm defined in SPEED and reliability is maintained by probabilistic multipath forwarding. DEED, a soft real-time communication protocol for sensor networks, considers both energy and end-to-end delay [59]. To do so, DEED builds a dynamic delay-constrained minimum-energy dissemination tree. Like RAP and SPEED, it also assumes location information for each node. A Real-Time Power-Aware Routing protocol (RPAR) is proposed in [60]. It makes routing decisions based on real-time performance and energy efficiency. RPAR assumes each nodes knows its location and is capable of dynamically adjusting its transmission power.

Chapter 4

Network Services Architecture

Directed diffusion serves as the baseline network protocol for our S&R system. We have developed three improvements for diffusion to lessen its weaknesses and enhance its strengths. We first describe in detail passive clustering for directed diffusion (PCDD). Next, we explain a route repair mechanism for directed diffusion which emphasizes localization of repair (LRDD). Thirdly, we outline the architecture of a real-time communication protocol for directed diffusion (RTDD) which improves the on-time delivery performance of diffusion. The network services improve efficiency, robustness, and timeliness of delivery for directed diffusion. More generally, the purpose of the enhanced communication services is to enable developers to easily utilize the power of the distributed sensor environment without its inherent complexities. The improved network communication mechanisms allow the system to function in challenging environments, which may have previously hindered functionality. PCDD reduces congestion resulting from network flooding. Flooding reduction is especially relevant in the context of energy efficiency and network latency. LRDD is crucial for reliable operation of the system in the face of node failures. It provides an efficient method of route healing to cope with node failure. Also critical to the performance of the system is timely response to detected events. To this end, RTDD performs packet prioritization and, thereby, reduces the number of packets that miss deadlines. RTDD is similar to RAP [56], but has been adapted to the two-phase pull model of directed diffusion. It has also been extended to support location-unaware networks. The distributed services, the lookup service, composition service, and adaptation service, may also be used in the S&R

architecture for greater system usability and reliability. Figure 4.1 shows a layered overview of the S&R architecture.

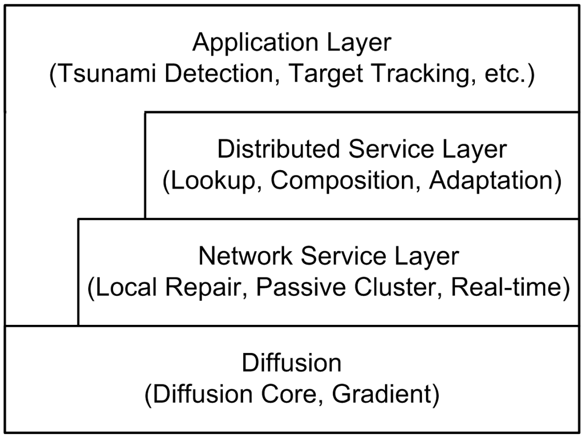
Application Layer
(Tsunami Detection, Target Tracking, etc.)
Distributed Service Layer
(Lookup, Composition, Adaptation)
Network Service Layer
(Local Repair, Passive Cluster, Real-time)
Diffusion
(Diffusion Core, Gradient)

Figure 4.1: Layered Architecture

Note that the distributed services reside above the proposed network services. The distributed services can take advantage of the lower level services provided by the network services (i.e., efficient flooding, route repair, and real-time communication). A more detailed illustration of the architecture of the system is shown in Figure 4.2. This diagram shows the interactions among the distributed services and among the network services. Also illustrated is the abstraction of the underlying routing protocol. Although applications and distributed services see diffusion as the routing entity, in actuality the diffusion core, gradient, and all three network services are working to deliver packets. Thus, the network services transparently benefit applications.

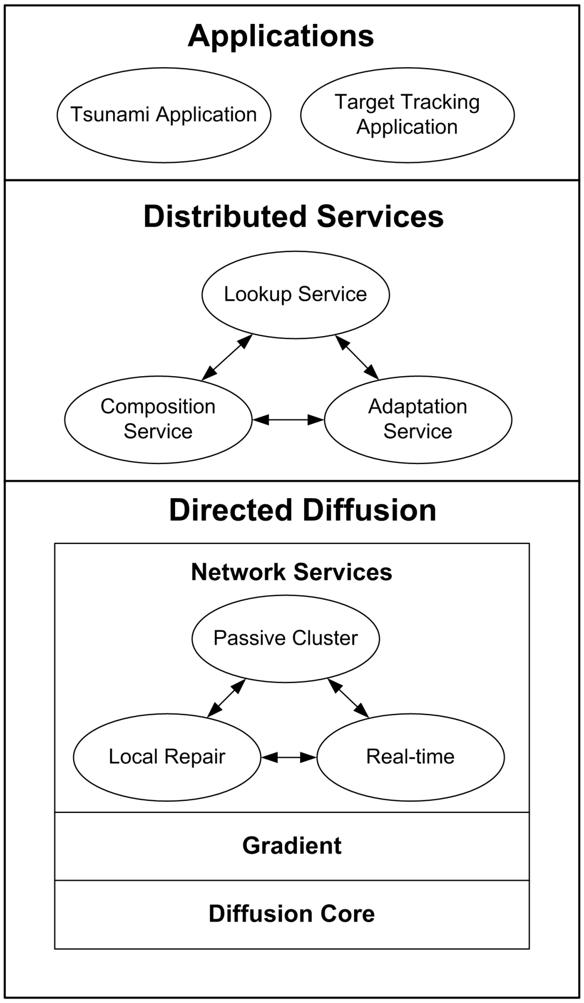

Figure 4.2: Detailed Architecture

4.1 Clustering Mechanism (PCDD)

As a result of the high cost of flooding and diffusion's strong reliance on it, we chose to implement a clustering mechanism for efficient flooding. We selected passive clustering (PC) for this purpose because its design objectives are very compatible with diffusion. In this section, we give an overview of passive clustering and describe the detailed workings of the protocol. Finally, we describe the minor modifications necessary to adapt passive clustering to diffusion, i.e., to create PCDD.

4.1.1 Passive Clustering Overview

The distinctive characteristic of passive clustering is its use of on-going data traffic to initiate cluster formation and communicate cluster-related information among the nodes. Using promiscuous packet reception, nodes gather cluster status information about all their 1-hop neighbors and adjust their own cluster state accordingly. Passive clustering creates clusters by assigning one of three states to a node. Nodes may be cluster head (CH), gateway (GW) or ordinary nodes (OR). CHs serve as leader nodes for their clusters and forward flooded packets to each member of the cluster. GWs connect two or more CHs together, thus serving as cluster relays. ORs receive flooded packets from CHs but do not forward the packet to their neighbors. Passive clustering results in a clustered structure similar to the topology shown in Figure 4.3. The circles represent cluster boundaries.

Corresponding to the three node states, PC defines three fundamental rules for operation: *first cluster head declaration wins*, *gateway selection heuristic*, and *ordinary nodes drop flooded packets*.

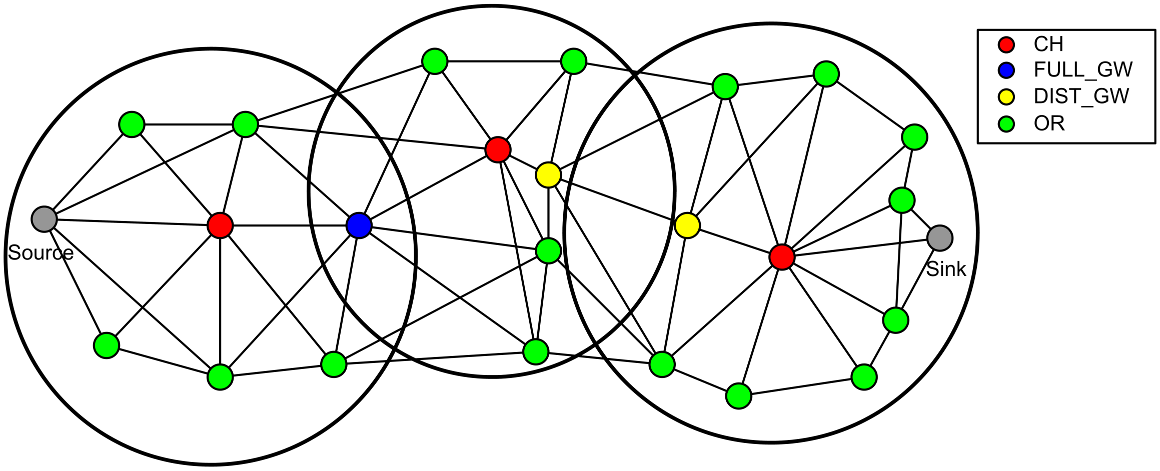

Figure 4.3: Example PC Topology

First Declaration Wins

Perhaps the most distinctive PC rule is its cluster head election rule. PC selects cluster heads by allowing the first node that declares itself CH to become CH. This is known as the *first declaration wins* rule. If a node has not heard from another CH, it claims itself to be CH and rules the rest of the nodes in its radio range. The first declaration wins rule provides several advantageous properties. Unlike many conventional clustering schemes, PC requires no waiting period or neighbor checking requirement to elect a CH. A node becomes CH as soon as it declares itself to be CH. The first declaration wins approach is also less likely to result in chain re-clustering [46]. In traditional clustering protocols, when two cluster heads move within transmission range of each other, one of them must defer to the other. This can trigger cluster head changes that propagate throughout the network [61]. The first declaration wins rule reduces this effect. Thirdly, PC creates a clustering that more closely resembles the data flow in the network. By creating cluster heads based on traffic, PC achieves a better correlation between traffic flow and resulting clustered topology.

Gateway Selection Heuristic

If too many nodes become gateways, the flooding efficiency decreases since such a large number of nodes forward flooded data. If too few nodes become gateways, network connectivity may be adversely affected. The goal is to choose a sufficient number of gateways to preserve connectivity, but very few more. To make this trade-off dynamically, PC defines a *gateway selection heuristic*. The gateway selection heuristic limits the number of nodes that become gateways without breaking the passive nature of PC. A node becomes a gateway according to the number of cluster heads and gateways it has overheard. Whenever a non-cluster head node hears a packet from a cluster head or gateway, the node becomes a gateway if Equation 4.1 is true. Otherwise, the node will become an ordinary node.

$$\alpha \cdot num(GW) + \beta > num(CH) \tag{4.1}$$

Note that in Equation 4.1, $num(GW)$ is the number of neighbors known to be gateways, $num(CH)$ is the number of neighbors known to be cluster heads, and α and β are tunable parameters ($\alpha, \beta \geq 0$). The values of α andβ should be chosen based on factors such as channel quality, noise level, and traffic patterns [5]. They may be locally adjusted to provide better adaptability and flexibility. This gateway selection procedure is fully distributed and requires only local information. It relies on overheard packets instead of active packet exchanges (e.g. cluster head-list exchanges). The disadvantage of this approach is that network connectivity may be affected for wrong values of the parameters. If the parameters are too aggressive in reducing the number of gateways, the topology may be partitioned.

Ordinary Nodes Drop Flooded Packets

The behavior of ordinary nodes lies at the heart of flooding reduction. A node that is neither a cluster head nor a gateway becomes an ordinary node. When a node identifies itself as an ordinary node, it no longer forwards flooded packets. All flooded packets received by an ordinary node can safely be dropped because neighboring nodes will have already received the packet (either from a CH or a GW). Hence, flooding efficiency is dependent on the number of ordinary nodes that can be found. In a sufficiently dense network, a relatively large number of ordinary nodes should be found.

4.1.2 Protocol Details

PC defines seven clustering states for nodes: two internal states and five external states. Internal states are entered when a packet is received and serve as a tentative role for the node while the packet is being processed. The two internal states are *Gateway-Ready* (GW_READY) and *Cluster Head-Ready* (CH_READY). Nodes enter external states when a packet is sent. These are externally visible states which are communicated to neighboring nodes and used in making adjustments of node state. The external states are *Initial* (IN), *Cluster Head* (CH), *Full Gateway* (FULL_GW), *Ordinary Node* (OR), and *Distributed Gateway* (DIST_GW).

Initial

On startup, nodes enter the *Initial* state. A node in *Initial* state does not belong to any cluster. Nodes move from the *Initial* state to one of the two internal states when a packet is received. PC maintains soft-state clusters using an implicit timeout scheme. If a node does not receives any packets within time interval $t_{clustertimeout}$, it reverts back to *Initial* state. Upon future reception of packets, the node will restart the clustering algorithm.

Cluster Head-Ready

Cluster Head-Ready is the internal state of potential cluster heads. A node in *Initial* state changes its state to CH_READY only when it has received a packet from a non-CH node. Simultaneous cluster head can sometimes occur due to topology changes and packet delay. To resolve these conflicts, PC uses a Lowest ID algorithm to select the node with the lowest ID to serve as cluster head. If a cluster head receives a packet from another cluster head with a lower ID, it gives up its role and enters the *Gateway-Ready* state. The node which loses the LID completion sends a CH Give-Up message to its neighbors informing them of the change in status.

Gateway-Ready

The *Gateway-Ready* state is the internal state of potential gateways (both full and distributed). A node enters GW_READY state when it receives a packet from a CH. A GW_READY node will become a gateway or an ordinary node depending on the gateway selection heuristic. A node will change from gateway-ready to FULL_GW or DIST_GW if an insufficient number of its neighbors are already gateways.

Cluster Head

The *Cluster Head* state is the external state of cluster heads. A node enters the CH state from the CH_READY state if it wins the Lowest ID competition or if it has not overheard any other cluster heads.

Full Gateway

Nodes in the state *Full Gateway* directly connect two cluster heads. A full gateway is thus a member of two clusters. Full gateways announce the IDs of the two CHs that they connect. A node that is reachable from two CHs may declare its role as FULL_GW only if it has not heard from another FULL_GW node announcing the same pair of IDs. In this way, gateways also follow the first declaration wins rule. If two nodes concurrently declare themselves as FULL_GW for the same pair of CHs, the node with the lowest ID will win and the loser will become an ordinary node. Full gateways are in the external state FULL_GW. Figure 4.4 illustrates a full gateway.

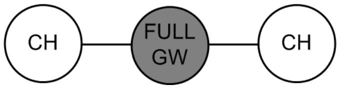

Figure 4.4: Full Gateway

Distributed Gateway

A node that is in the *Distributed Gateway* state connects two clusters, but is not directly connected to one of the cluster heads. A distributed gateway is composed of two nodes working together to serve as a gateway between two clusters. This allows cluster heads that are two hops apart to be connected. Figure 4.5 illustrates a distributed gateway. Notice both the full and distributed gateways in the example topology in Figure 4.3.

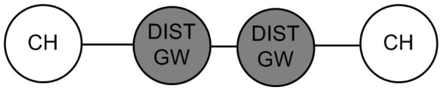

Figure 4.5: Distributed Gateway

Ordinary Node

Ordinary nodes are nodes that do not forward packets. A node enters the OR state from the GW_READY state based on the gateway selection heuristic. A node changes from GW_READY to OR if enough of its neighbors are gateways as determined by the gateway selection heuristic. When a node can hear from more than two CHs and every pair of announced CHs is connected by a FULL_GW it will become an OR.

Incoming Packet Processing

The transition state diagram shown in Figure 4.6 summarizes the passive clustering algorithm. When a packet is received at a node, the state of the node will be changed to one of the two internal states: CH_READY or GW_READY. An Initial node will change its state to CH_READY if the packet is from a non-CH (1). If the packet is from a gateway or an initial node, the node will enter the GW_READY state (5). CHs revert to CH_READY (3) and GWs (9) and ORs (8) revert to GW_READY upon each incoming packet. The receiving node always updates its neighbor lists based on the state of the sending node which is contained in the message. By stepping back to the GW_READY state (8, 9, and 11), nodes can adjust their state dynamically. For example, if the number of gateways has changed, an ordinary node may promote itself to gateway. When the soft-state of PC expires, nodes in each state revert back to Initial (12, 13, 14, and 15). As a result, new clusters may be composed in response to subsequent packet flooding. This potential for re-structuring is beneficial for sensor networks since cluster head responsibilities will be handled by different nodes throughout the life of the system, thus distributing energy consumption.

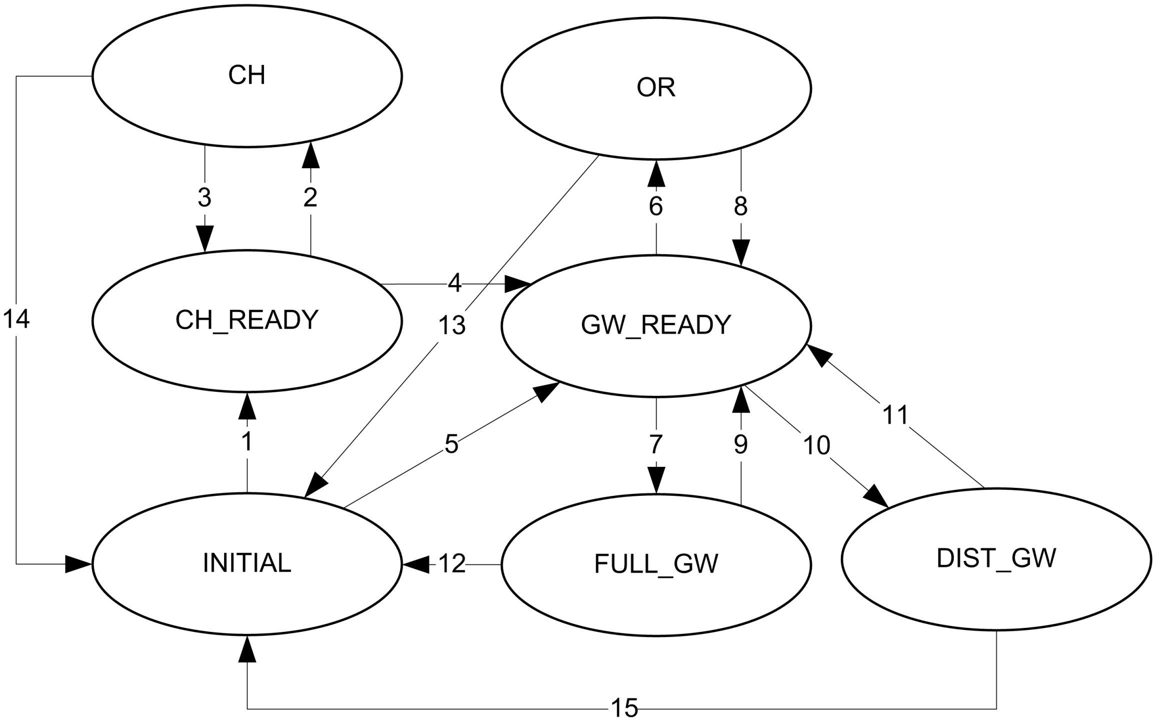

Figure 4.6: PC Transition State Diagram

Outgoing Packet Processing

When a packet is ready to be sent, the node changes its PC state from an internal state to one of the five external states. If it has not heard from any other CHs, a candidate

CH (CH_READY) will change its state to CH (2) when it has packets to send. Otherwise it will change to GW_READY. Nodes in the GW_READY state will become FULL_GW, DIST_GW, or OR. If the number of known CHs is greater than one and there is no gateway connecting any two CHs, the node becomes a FULL_GW (7). If a DIST_GW has announced another cluster not known by the current node and no FULL_GW connects them, then the node becomes a DIST_GW (10). Otherwise, it becomes OR (6). If the number of known CHs equals one, then the node becomes either a DIST_GW or an OR. If a node hears a DIST_GW announce a CH other than its CH or if there is no DIST_GW in the cluster, it will become a DIST_GW (10). Otherwise, the node will become an ordinary node (6).

4.1.3 Adaptation to Diffusion

PC state information must be appended to all flooded packets. To apply PC on a diffusion network, the state information is added to interests and exploratory data. Recall that diffusion creates gradients and periodically refreshes them. After this initial setup phase, flooding is no longer necessary since data flows over reinforced paths. Since PC piggybacks cluster status information only on flooded packets, PC creates the clustered structure during the initial setup phase of the gradient cycle. For the remainder of the cycle, no PC state information is transmitted, so PC is essentially inactive. PC is only active during period gradient refresh when diffusion floods the network with interests and exploratory data.

Since PC affects the route setup, it is possible for suboptimal routes to be discovered. The structured topology created by PC may exclude the optimal route between source and sink. This end-to-end route from source to sink will only include nodes identified to be cluster heads and gateways – no ordinary nodes will be on this path. This potential for suboptimal routes is one of the inherent trade offs of PC as compared to global flooding. PC will improve flooding efficiency but may result in slightly longer routes.

4.2 Repair Mechanism (LRDD)

To deal with the shortcomings in diffusion's ability to adapt to failure and mobility, we have developed a mechanism to efficiently handle route repair. We call our local repair protocol for directed diffusion LRDD. Our solution emphasizes truly localized repair in order to reduce latency and energy expenditure. Its basic structure is similar to the local repair algorithm used in ADMR [47], but its methods noticeably differ from ADMR since it is tailored to directed diffusion. We divide the local repair problem into three phases: break detection, break localization, and localized gradient repair. We will describe how LRDD handles each of these phases in this section.

4.2.1 Break Detection

The first step in adapting to node failure or mobility is detecting the link breakage. This may be accomplished in a variety of ways. We assume that appropriate algorithms may be used to reliably detect a link breakage. Our focus is on handling the break after it is detected not adaptively detecting path breakage. For our experiments, we used a fixed event rate known a priori to detect breaks. We identified a link as broken when no data was received from a flow after an entire event interval had elapsed.

4.2.2 Break Localization

Once a break has been detected, the break localization phase begins. The goal of this phase is to identify the node immediately downstream from the broken path. This node will initiate the localized gradient repair algorithm described in the next section. All intermediate nodes that detect a break will send a repair notification message to their 1-hop downstream neighbors along the gradients for the missing data. Every intermediate node downstream from the break should send a repair notification, and every intermediate node except the one nearest to the break should also receive a repair notification. If a node does not receive a repair notification within t_{repair} seconds after sending one, it must be the nearest node, so it initiates the localized gradient repair.

4.2.3 Localized Gradient Repair

The heart of LRDD is the localized gradient repair phase. The goal of this phase is to find and create new gradients in the area near the break by using the same basic mechanisms as global gradient repair. We first describe several potential mechanisms for restricting flooding to a localized region and then explain the inner workings of the repair process itself. Figure 4.7 illustrates the break localization and localized gradient repair phases of the algorithm.

Local Flooding

In order to restrict the flooding required to find new paths, we limit the packet forwarding in one of several ways. Possible methods include simple hop-limited flooding or

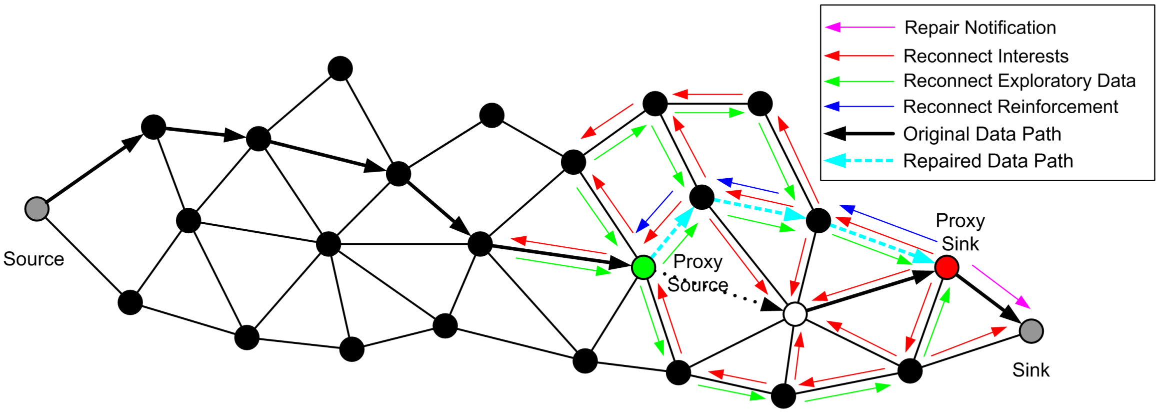

Figure 4.7: Local Repair for Directed Diffusion

limiting the reconnect packets to the 1-hop neighbors of the failed node. The most common approach to localized flooding among repair protocols is hop-limited flooding. In this approach, a hop count (or time to live) field is incremented on each hop. When the hop count exceeds a threshold value, the flooded packets are dropped. Another straightforward approach is to limit the flooding to 1-hop neighbors of the failed node. This can be easily implemented by including the ID of the failed node on the repair packets and restricting flooding to nodes who have overheard packets from that failed node. A third and more sophisticated strategy is to limit flooding to nodes in the one or two clusters around the failed node.

Reconnect Interests

The first step in localized gradient repair is the local flooding of *reconnect interest* messages. These interest messages for the broken data flow are essentially searching for nodes upstream from the break which are still receiving data. Reconnect interests are only transmitted in a limited neighborhood (as defined by the localized flooding algorithm) around the node nearest to the break (identified during the localization phase). This originator node acts like the sink in global gradient repair. For this reason, we label it the *proxy sink* in local gradient repair. Nodes outside the area determined by the local flooding algorithm will drop reconnect interests, enforcing the region boundaries. Reconnect interests are flooded in the region near the break in search of upstream nodes still connected to the dataflow.

Reconnect Exploratory Data

In response to reconnect interests, upstream nodes on the data path that are still receiving data transmit *reconnect exploratory data* messages to neighbors that send reconnect interests. Reconnect exploratory data is exploratory data that is sent back to the proxy sink. In order to perform the most localized repair, the node directly upstream from the break should be found. This node will act like the source in global gradient repair, so we call it the *proxy source* in local gradient repair. To achieve this behavior, reconnect exploratory data messages are only sent from nodes that have not overheard reconnect exploratory data. Thus, the first node to send reconnect exploratory data will become the proxy source. In summary, only nodes along the original path can send reconnect exploratory data. The node nearest upstream to the break should be identified as the proxy source since it should be the first node along the existing path to receive reconnect interests. LRDD will still function if another node further upstream from the break is identified as the proxy source.

The proxy source will send reconnect exploratory data back to the interest initiator (the proxy sink).

Reconnect Reinforcement

In global gradient repair, the sink reinforces the fastest path over which the exploratory data is received. Correspondingly, in the case of local repair, the proxy sink sends *reconnect reinforcement* messages to the first neighbor that delivers a reconnect exploratory data. In turn, this neighbor node will reinforce its fastest neighbor all the way back to the proxy source. While this path is probably not optimal, it serves as a temporary fix until the next global gradient refresh. Since the search for the new path was restricted to a relatively small region, it is almost identical to the original optimal path found by diffusion except for a few hops where the repair was performed. As a result, the repaired path should provide a reasonably low-latency route from source to sink.

4.3 Real-Time Communication Mechanism (RTDD)

To support timely communication, we have developed a real-time communication service for directed diffusion similar to RAP [56]. Recall from Section 3.5.1 that the RAP architecture defines a five-layer network stack. Many of the layers in the RAP network stack are handled natively by diffusion. The top layer of RAP, a query-event service that allows events to be registered and triggered, is inherently provided by the data-centric, publish-subscribe nature of diffusion. Diffusion operates without a transport layer, so RAP's Location-Addressed Protocol is unnecessary in diffusion. Diffusion also handles the responsibilities assigned to Geographic Forwarding in RAP since diffusion routing is based on attribute vectors. Although routes in diffusion are more costly to establish due to global flooding, no location information is required. RAP, in contrast, requires each node to know its own location requiring costly GPS hardware on every node. The core component of RAP is Velocity Monotonic Scheduling (VMS), the distance-aware and deadline-aware scheduling policy. VMS prioritizes messages by computing a velocity based on the packet's distance to its destination and its temporal deadlines. Although packets are prioritized at the MAC and network levels in RAP, our implementation only performs prioritization at the network layer. While this may decrease the ability of RTDD to prioritize packets, it provides a clean separation of layers (i.e., RTDD, a network-layer protocol does not require changes in the MAC layer).

The two-phase pull model of directed diffusion may be easily extended to support real-time data flows. The primary additions to diffusion required for RTDD are a prioritized queue and a scheduling policy. We have developed both static and dynamic scheduling policies for RTDD equivalent to SVM and DVM in RAP. In addition, we have extended the protocol to compute priority without requiring each node to possess location information. We call these protocols Static Absolute Time (SAT) and Dynamic Absolute Time (DAT) since they are based on absolute time. We have also developed protocols based on relative time differences: Static Relative Time (SRT) and Dynamic Relative Time (DRT). The advantage of using relative time is a decreased dependence on global time synchronization. RTDD can use any of the six algorithms for computing priority: SVM, DVM, SAT, DAT, SRT, or DRT. If location information is known by each node, SVM or DVM may be used to prioritize packets. Otherwise, one of the time-based protocols will be utilized.

4.3.1 SVM and DVM

To provide support for deadlines, RTDD simply adds a few new attributes to a data flow. Before interests are flooded from the sink node, the location of the sink is added to the packet as a new attribute/value tuple. Exploratory data then flows back to the sink from the source. Next, reinforcements are sent along the fastest path between sink and source. As the data is published at the source, RTDD computes the priority of the packet based on the deadline (supplied by the application) and the location of the sink (supplied by the interest packet). The priority is computed as the distance between the source and sink divided by the deadline (Equation 4.2).

$$V = \frac{\|p_{src} - p_{dst}\|}{T_{deadline}} \tag{4.2}$$

For DVM, this value is updated at each intermediate hop based on the current progress of the packet (Equation 4.3).

$$V = \frac{\|p_i - p_{dst}\|}{T_{deadline} - T_i} \tag{4.3}$$

To enable this dynamic calculation, the source must timestamp the data packets before they are transmitted. This allows the intermediate nodes to compute the time expired since the packet was sent (T_i in Equation 4.3). Packets are then queued in priority order at each hop and retransmitted in prioritized order. Note that DVM requires intermediate nodes to know their location so that the updated priority may be computed. Also notice that DVM assumes global time synchronization in order to compute time differences at each intermediate node.

4.3.2 SAT and DAT

If location information is not available, a time-based approach is used to estimate the distance from source to sink. Instead of appending location information to interest messages, nodes add timestamps to the packets. Since diffusion creates routes using a two-phase packet exchange, the time delay between source and sink can be estimated without introducing significant overhead. When reinforcement messages are received at the source, the time delay between the source sending the exploratory data and the sink sending the reinforcement message is computed. We assume this time difference is proportional to the distance from source to sink. Note that this approach also assumes time synchronization among the nodes. Several time synchronization protocols for sensor networks have been proposed [62] [63] [64] [65], so this requirement is not infeasible. Like the location-based protocols, two versions of the absolute-time-based algorithm are also defined. The priority may be calculated statically at the source (SAT) or dynamically at each hop (DAT) based on absolute time. In the former case, the priority is computed according to Equation 4.4.

$$P = \frac{t_{sink} - t_{source}}{T_{deadline}} \tag{4.4}$$

In this equation, t_{source} is the time the exploratory data packet was sent from the source, t_{sink} is the time the reinforcement message was sent from the sink, and $T_{deadline}$ is the deadline of the data flow (in units of time, not a timestamp). The time-based protocols compute a priority value that is a ratio of times.

In the dynamic case, the priority of a packet is re-calculated at each hop. If the source timestamps exploratory data messages and the sink timestamps reinforcement messages, then each node along the reinforced path can calculate the time delay from itself to the sink. Each node along the data path must cache the delay between the most recent exploratory data and the reinforcement message in order to support distance awareness. Data messages must also be timestamped by the source so that intermediate nodes can calculate elapsed time for a given packet. The elapsed time is subtracted from the deadline to gauge the deadline urgency of the message. DAT calculates this priority value using Equation 4.5.

$$P = \frac{t_{sink} - t_i}{T_{deadline} - T_{elapsed,i}} \tag{4.5}$$

In this case, t_i represents the time the exploratory data packet was sent from the intermediate node i, t_{sink} is the time the reinforcement message was sent from the sink, $T_{deadline}$ is the deadline of the data flow, and $T_{elapsed,i}$ is the time elapsed in sending the data packet to hop i. Elapsed time, $T_{elapsed,i}$, is computed as $T_{elapsed,i} = t_{now} - t_{data}$ where t_{now} is the current time and t_{data} is the time the data packet was sent from the source. To simplify the protocol, we estimate the delay between intermediate node and sink (the numerator of Equation 4.5) as the total end-to-end delay minus the elapsed time, or more formally, $t_{sink} - t_i \approx t_{sink} - t_{source} - T_{elapsed,i}$. Thus, Equation 4.5 becomes

$$P = \frac{t_{sink} - t_{source} - T_{elapsed,i}}{T_{deadline} - T_{elapsed,i}} \tag{4.6}$$

4.3.3 SRT and DRT

As an improvement upon SAT/DAT, we have also developed variants of RTDD which compute priorities based on relative time differences. The static and dynamic versions of the relative time difference protocols are SRT and DRT. The primary advantage of using relative time is the reduced dependency on time synchronization. SRT and DRT use the round trip time as a measure of path length (as opposed to end-to-end delay in SAT/DAT). In SRT, the source stores timestamps when it sends exploratory data and when it receives the reinforcement message. Since both time measurements are taken at the source, global time synchronization is not be necessary. Similar to SAT, the priority of packets in SRT is computed statically at the source as the quotient of the round-trip delay and the deadline as shown in Equation 4.7.

$$P = \frac{t_{reinforcement,\ source} - t_{exp.\ data,\ source}}{T_{deadline}} \tag{4.7}$$

DRT computes the priority dynamically at each hop based on the round-trip time and the elapsed time. At hop i, the priority is computed according to Equation 4.8

$$P = \frac{(t_{reinforcement,\ source} - t_{exp.\ data,\ source}) - T_{elapsed,i}}{T_{deadline} - T_{elapsed,i}} \tag{4.8}$$

Note that in Equations 4.7 and 4.8, $t_{reinforcement,\ source} - t_{exp.\ data,\ source}$ represents the time between when the source sends exploratory data and when it receives a reinforcement, i.e., the round trip time. Again, also note that all the dynamic versions of RTDD (DVM, DAT, and DRT) require time synchronization in order to calculate the elapsed time.

The time-based protocols significantly enhance the applicability of RTDD for various network environments. The time-based techniques achieve the same goal, packet prioritization based on both distance and deadline, but do so without the strong localization requirement of SVM and DVM. The numerators in Equations 4.4 - 4.8 correlate to distance awareness, and the denominators encapsulate deadline awareness. Although our location-free design requires more communication overhead than the location-based algorithms, the communication is essentially free since the timestamp information is piggybacked on the packets involved in diffusion's two-phase path discovery protocol. No additional packets are required, only a slightly increased packet size. This trade-off may be advantageous for applications where location information is not available but time synchronization is possible.

Chapter 5

Network Services Design

5.1 Design Principles

In designing the network services, we were guided by several principles. Our primary design goals were energy-efficiency, scalability, localization, distributability, real-time communication, and reactivity. Notice that our design principles are strongly correlated to the strengths of directed diffusion. In this section, we describe each of these design goals and explain their implications on our design decisions.

5.1.1 Energy-efficient

Power consumption is of paramount importance in sensor networks since devices usually cannot easily be recharged. Consequently, our design incorporates several features that promote energy-efficiency. The primary purpose of PCDD and the local flooding involved in LRDD is to save energy by minimizing communication. Communication is the primary energy expense in sensor networks, so reduced transmissions equate to saved energy. The energy required to transmit one bit is several orders of magnitude larger than the energy needed to perform one operation. According to [66] one ground to ground transmission of 1 kb over 100 m expends as much energy as processing 300 million instructions. Hence, reducing unnecessary communication is essential for sensor networks. PCDD accomplishes this by reducing the number of messages involved in route creation. LRDD reduces the number of messages required for route maintenance. The design of RTDD also emphasizes energy-efficiency in that no new communication overhead is required. Data for setting up the real-time protocol is piggybacked on top of the standard two-way message exchange used by diffusion for route discovery/refresh. Thus, all three network services work toward the goal of energy-efficiency.

5.1.2 Scalable

Another general goal of our system and all sensor networks is scalability. Protocols should scale up to networks with large numbers of nodes. The reliance of standard directed diffusion on global flooding greatly limits the scalability of the protocol since flooding is so costly for large networks. PCDD and LRDD both address the goal of scalability by reducing unnecessary transmissions. As the network size increases, these unneeded communications lead to network congestion. In sufficiently large networks, flooding-induced congestion can completely cripple the network. Hence, efficient flooding is vital to the robust operation of large-scale networks. The flooding reduction mechanisms incorporated into PCDD and LRDD support the goal of scalability.

5.1.3 Localized

Localized protocols maintain information about one-hop neighbors. This significantly reduces the amount of state information that must be stored and hence, improves the scalability and simplicity of the algorithm. By decreasing the complexity of an algorithm, the principle of localization simplifies the design and implementation. One of the best properties of directed diffusion is its completely localized nature. Since our network services are deeply integrated into diffusion, they by design complement its localized aspects. Specifically, PCDD exhibits this localized nature in that nodes choose their state based only upon the states of their one-hop neighbors; two-hop neighbor information is not required. This avoids the continual exchange of Hello messages, greatly reducing the communication overhead of PCDD compared to other efficient flooding algorithms. LRDD also supports the goal of localized interactions by localizing its repair work to the area immediately surrounding the failed node. Thus, the advantages of localized interactions are preserved by our algorithms.

5.1.4 Distributed

The distributed nature of sensor networks is one of their most challenging and powerful characteristics. The goal is to fully distribute algorithms over all the nodes in the network, in contrast to typical algorithms which utilize the client/server model. Our protocols address this challenge by distributing tasks among all the nodes. This is particularly evident in PCDD where the clustered structure is created dynamically according to the *first declaration wins* rule. Unlike many traditional algorithms, cluster formation in PCDD is completely distributed. Similarly, in LRDD, any node may potentially detect and repair a breakage. In this sense, all nodes act as adaptation servers forming a completely distributed adaptation service. LRDD distributes the responsibility for repair among all the nodes. The design of our protocols emphasizes distributed operation.

5.1.5 Real-Time

Timely communication is an important goal for many applications. RTDD aims to achieve the goal of real-time communication. The prioritized queue gives preference to more urgent packets, helping them meet their deadlines. RTDD supports hard deadlines by dropping packets when their deadline has been exceeded. This further reduces congestion and gives other packets a greater chance of meeting their deadlines. RTDD significantly enhances the capabilities of directed diffusion, allowing the network to support time-critical data flows.

5.1.6 Reactive

Our final design goal is to develop reactive protocols and mechanisms. In general, reactive protocols are more suitable to sensor networks than proactive protocols since they are more energy efficient. Reactive protocols respond to events instead of proactively maintaining state information ahead of time. In energy-constrained systems, it makes little sense to continuously maintain routes if no data is being transferred over those routes. This is especially relevant in sensor systems which infrequently detect events of interest.

PCDD and LRDD particularly incorporate reactive features in their designs. PCDD reactively creates clusters in response to and through the use of ongoing packet transmissions. Since packets are not exchanged beforehand, no energy is consumed until it is necessary. LRDD responds to repair broken links by reactively repairing them. Unlike standard directed diffusion which periodically repairs broken links, LRDD is fundamentally a reactive technique. In actuality, LRDD is a reactive version of diffusion's own route discovery algorithm but is performed locally instead of globally.

5.2 Design Decisions

5.2.1 Clustering Mechanism (PCDD)

The compatibility of passive clustering and diffusion was recognized by Handziski et. al [67] who first implemented passive clustering over directed diffusion. Passive clustering is especially well-suited to sensor networks due to its localized, reactive, and fully distributed nature. All state changes are made using local knowledge gained by overhearing neighboring nodes. Passive clustering's reactive cluster formation is well-matched to diffusion's reactive publish-subscribe model. Furthermore, the fully distributed nature of passive clustering maps nicely to any sensor network routing protocol, especially diffusion. No client/server message exchanges are needed. Passive clustering does not need periodic messages, but instead takes advantage of existing packets. Finally, the protocol is very resource-efficient regardless of the size or density of the network.

5.2.2 Repair Mechanism (LRDD)

Break Detection

While break detection was not the focus of our research, we considered several potential methods for detecting a broken link. One approach is to calculate an expected time for data events to be received from each neighbor. The source may monitor its output and calculate an outgoing event rate for each gradient. It will then append this event rate to each data packet. Intermediate nodes need only to store the data rate of the packet and the time it was received. If the interval has been exceeded, a break will be detected. Another strategy for break detection is for intermediate nodes to calculate expected receive times based on incoming packet reception rates. Event intervals are calculated by monitoring the input from each neighbor at intermediate nodes. The advantage of this approach is that it is completely localized and distributed. Another possibility is to use physical layer metrics (e.g. signal strength, or SNR) to detect failing links. Given event rate T_r, we detected link breakage when a packet was not received in $1.5 \cdot T_r$. More complex event interval determination algorithms should be investigated for production systems.

Break Localization

Break localization may be carried out differently depending on the break detection scheme used. If physical layer metrics are used to identify breakages, then our break localization scheme may be completely unnecessary. Our design assumes no access to physical layer information and no explicit network layer acknowledgments. Instead, our scheme utilizes an implicit timeout procedure to detect link failure. The advantage of this approach is that it may be broadly used regardless of physical or MAC layer differences. Since no physical or MAC layer assumptions are made, our repair protocol will support a wide variety of architectures.

Local Gradient Repair

In order to restrict the flooding required to find new paths during local gradient repair, we restrict packet forwarding in one of several ways. Flooding boundaries may be found using the clusters created through the passive clustering protocol, a hop-limited flood, or the failed node ID. These mechanisms restrict the flooding of interest and exploratory data to a limited area.

Our design separates the local flooding algorithm from LRDD. By decoupling the local flooding mechanism from the repair protocol, we gain several benefits. First, from a software engineering perspective, the modularized software is easier to write, debug, and maintain. Secondly, our repair algorithm can be paired with any local flooding strategy. Thus, our design may easily be extended by more complex local flooding algorithms. Thirdly, the decoupling allows multiple local flooding algorithms to be used simultaneously. Multiple algorithms could be stacked on top of each other or used in different regions of the same network. For example, an algorithm could be adaptively chosen based on local conditions, e.g., network density or congestion.

5.2.3 Real-time Communication Mechanism (RTDD)

The prioritized queue is the essential component of RTDD. Implementing it proved to be a challenging task because several different approaches were possible. We recognized three options for the implementation of a prioritized queue in the diffusion API.

1. Delay-Based: Set event delay times in proportion to priority levels.
2. Block-Based: Prioritize all events that have expired during the receiving period.
3. Timer-Based: Send the highest priority packet after the expiration of a recurring send timer.

The delay-based scheme uses priority to compute the delay time associated with a particular packet. The delay time is inversely proportional to the priority. Hence, high priority packets have a lower delay while low priority packets are assigned a longer delay time. The second option is to prioritize blocks of packets that have expired timers. Diffusion switches from receiving and sending when there are no packets to receive. It then processes all expired timers, i.e., all packets in the event queue with expired send times. This block of packets is usually sent in the order it was received. Using the block-based approach, the block of expired packets will be sent in priority order. Finally, the timer-based approach uses a timer to repeatedly send one packet every t_{timer} milliseconds. When packets are received, they are added to a queue in priority order. In a separate thread, a recurring timer sends the highest priority packet each time the timer expires.

The delay-based approach has one major drawback – it delays low priority packets when there are no high priority packets to be sent. Hence, low priority packets are penalized unnecessarily and experience greater latency as a result. The block-based scheme also presents significant difficulties. It prioritizes within very small blocks of packets because the network so frequently switches between send and receive mode. This leads to a very small granularity of prioritization which may be almost imperceptible to the application. The prioritization occurs at such a minute level that it provides no benefit whatsoever. Because of these problems, we chose to implement the timer-based algorithm. While this results in the addition of a timeout parameter, it is superior to the other two approaches since it effectively prioritizes packets without unnecessarily delaying low priority packets. We describe the implementation this approach in more detail in Section 6.3.3.

The time-based protocols compute priority based on the deadline as given by the application and the distance from source to sink as estimated by communication delay. Although several message exchanges occur in diffusion route creation, we choose to measure the time delay of the exploratory data message as it is sent from source to sink. We chose this message transmission because it follows the same path and direction as the actual data.

DAT and DRT re-calculate the priority of a packet at each intermediate node based on the elapsed time. As the most complicated cases, they posed the most design options. The primary problem was in calculating the time to the sink (the numerators in Equation 4.5 and 4.8). We considered two approaches: stateful and stateless. In the stateful approach each intermediate node along the path from sink to source must maintain a time value for every data flow passing through it. The value may be computed by storing $t_{now} - t_{sink}$ for each reinforcement message received. This represents the time from node i to the sink. These values should also be indexed by data flow. In the stateless approach, all the information needed to compute priority is stored within each data packet. Using this strategy, the time to the sink is estimated by subtracting the elapsed time from the total end-to-end time, $t_{now} - t_{sink} - T_{elapsed}$. This gives an approximation for the time from the intermediate node to the sink. The stateful approach is more complex and requires greater storage requirements per node. However, it provides a better estimate of the time to the sink. The stateless algorithm is simpler to implement and more scalable but may be less accurate. We chose the stateless approach due to its simplicity and scalability. It provides sufficient accuracy, especially over longer durations.

Chapter 6

Network Services Implementation

6.1 Clustering Mechanism (PCDD)

We implemented passive clustering using the Filter API provided by ISI diffusion [68]. This required the creation of a new diffusion attribute and a program with two filters. The attribute was used to relay information about the passive clustering state of each node to neighboring nodes. The filters intercept packets and update state information based on the PC state of the previous hop.

6.1.1 Passive Clustering Attribute

In order to communicate information about the passive clustering state of each node, we defined the class `PCInfo_t` to encapsulate all the passive clustering status information including state, node ID, and the cluster information. The `PCInfo_t` class is defined and explained below.

```
class PCInfo_t \{
    int nodeID;
    int state;
    int CH1;
    int CH2;
    timeval ts;

    void fromAttr(NRSimpleAttribute<void *> *attr)
};
```

- `nodeID` - The diffusion ID of the node

- state - An integer representing the external passive clustering state of the node
- CH1 - The ID of the node's primary cluster head
- CH2 - The ID of the node's secondary cluster head; For CH and OR nodes, CH2=-1
- ts - A timestamp corresponding to the last received message from the node
- void fromAttr(NRSimpleAttribute<void *> *attr) - Converts attribute attr to a PCInfo_t object.

To transport the PC information in diffusion packets, we pack the PCInfo_t object into a PC attribute and push the attribute onto the attribute vector of the message. The PC attribute is defined as follows:

```
#define PC_STATE_KEY  5000
NRSimpleAttributeFactory<void *> PCAttrFactory(PC_STATE_KEY,
                                               NRAttribute::BLOB_TYPE);
```

Upon reception of packets with this attribute, nodes unpack the data into a PCInfo_t object using its fromAttr() method. Nodes maintain a table containing PC information about each of their neighbors. The neighbor state table is updated on the reception of every flooded packet. The neighbor table is implemented in the class PCNodeList as shown below. The PCNodeList class stores information about the passive clustering state of neighboring nodes as an STL list of PCInfo_t objects. The core functionality provided by the class includes adding/updating neighbors as well as counting the number of neighbors in a particular state. The PCNodeList class also has helper methods that analyze the neighbor list to determine appropriate state changes.

```
class PCNodeList{
    list<PCInfo_t> neighbors;

    /* Core methods */
    void update(int nodeID, PCInfo_t pcInfo);
    int  count(int state);

    /* Helper method */
    bool anyTwoCHsNotConnected(int &ch1, int &ch2);
};
```

- `update(int nodeID, PCInfo_t pcInfo)` - Adds/updates PC information of node `nodeID` with PC status `pcInfo` to the neighbor list
- `count(int state)` - Returns the number of nodes in state `state` in the neighbor list
- `anyTwoCHsNotConnected(int &ch1, int &ch2)` - Returns true if any pair of known CHs are not connected by a GW and false if all pairs are connected. If a disconnected pair exists, `ch1` and `ch2` are set to their IDs.

6.1.2 Passive Clustering Filters

The PCInfo_t and PCNodeList classes are extensively used in the implementation of the passive clustering filter program. Our implementation utilized two filters in one program to intercept messages before and after the gradient filter. In the pre-gradient filter, nodes update their internal state to CH_READY or GW_READY according to the state of their neighbors. If no other CHs have been overheard, a node will enter CH_READY. Otherwise, it will enter GW_READY. Algorithm 1 summarizes the internal state update process that is executed when packets are received.

Algorithm 1: Incoming Packet Processing for Passive Clustering

```
Input: Message, Neighbor State Array, MyState
Output: Internal State
switch MyState do
    case INITIAL
        if $num(CH) = 0$ then
            InternalState = CH_READY
        else
            InternalState = GW_READY
        break
    case CH
        if $Message.State != CH$ then
            InternalState = CH_READY
        else
            if $myID < Message.ID$ then
                InternalState = CH_READY
            else
                InternalState = GW_READY

        break
    case FULL_GW
        InternalState = GW_READY
        break
    case CH_READY
        InternalState = GW_READY
        break

end
```

In the post-gradient filter, the external state of the node is determined and added to the outgoing packet. Nodes in CH_READY become CHs if they have not heard from any other CHs. Nodes in GW_READY enter FULL_GW or OR depending on the number of CHs and GWs among their neighbors. Algorithm 2 concisely summarizes the outgoing packet processing.

Algorithm 2: Outgoing Packet Processing for Passive Clustering

```
Input: Internal State, Neighbor State Array
Output: External State
if InternalState = CH_READY then
    if num(CH) = 0 then
        ExternalState = CH
    else if InternalState = GW_READY then
        InternalState = GW_READY
if InternalState = GW_READY then
    if num(CH) > 1 then
        if Any two CHs are not connected by any known gateway then
            ExternalState = FULL_GW
    else
        if $\alpha \cdot numCH + \beta < numGW$ then
            ExternalState = FULL_GW
        else
            ExternalState = OR
```

If the external state has been determined to be OR, then the interest or exploratory data packet will be dropped. This is accomplished in the Filter API by simply not forwarding the message back to the diffusion core. Thus, only nodes in the state CH or FULL_GW are allowed to forward interests and exploratory data. Nodes in OR will receive and process packets, but not forward them. Note that our implementation does not utilize distributed gateways. This simplifies the protocol without significantly harming its performance.

6.2 Repair Mechanism (LRDD)

The path repair mechanism, LRDD, is implemented using new attributes and the Filter API. LRDD is one program, but the local flooding mechanism is implemented separately.

This decoupling allows any local flooding algorithm to be used in combination with the core repair protocol. In this section, we describe the attributes used by LRDD and explain the details of its implementation.

6.2.1 LRDD Attribute

We define two new attributes that correspond to the new messages introduced by our protocol. The `Repair Notification` attribute marks a data packet as a repair notification. The `Reconnect` attribute is added to reconnect interests and reconnect exploratory data to differentiate them from standard route discovery packets created by diffusion. The new attributes are listed below.

```
#define REPAIR_NOTIFICATION_KEY       4500
#define RECONNECT_KEY                 4501

NRSimpleAttributeFactory<char *> RepairNotificationAttr(
                 REPAIR_NOTIFICATION_KEY, NRAttribute::STRING_TYPE);
NRSimpleAttributeFactory<int> ReconnectAttr(RECONNECT_KEY,
                                           NRAttribute::INT32_TYPE);
```

6.2.2 LRDD Filters

The majority of local repair is implemented in one program with two filters: a pre-gradient filter and a post-gradient filter. The pre-gradient filter handles the bulk of the work. It sets data timeouts after each data packet is received. If the data timeout expires without receiving a new data packet, repair notifications are sent one-hop downstream, and a repair notification timer is set. If it expires and no repair notification is received, the node becomes the proxy sink and floods interest messages with a reconnect attribute set to the failed node ID. The post-gradient filter maintains a list of upstream neighbors and downstream neighbors in order to detect link breakages. This filter also drops reconnect exploratory data at the proxy sink and reconnect reinforcements at the proxy source.

Interests and exploratory data are normally created by the diffusion routing (`dr`) object. To support reactive repair we added the following two new methods to the `dr` class:

- `int reconnectPublish(NRAttrVec *attrs)`
- `int reconnectSubscribe(NRAttrVec *attrs)`

These methods allow us to send reconnect packets when a break is detected. The LRDD filter at the proxy sink invokes `reconnectSubscribe()` to send reconnect interests, and the proxy source invokes `reconnectPublish()` to transmit reconnect exploratory data in response to reconnect interests. The proxy sink sends a reconnect reinforcement message to the first neighbor that sent it a reconnect exploratory data message. Algorithm 3 summarizes the logic involved in the pre-gradient filter.

Algorithm 3: Pre-Gradient Processing for LRDD

```
if Repair Notification then
    Update State as Not Proxy Sink
else if Data Message then
    Update Receive Data Time
    Reset Expected Data Timer
else if Reconnect Interest then
    if On Original Data Path and Still Receiving Data then
        Become Proxy Source
        Flood Reconnect Exploratory Data
    else
        Forward Reconnect Interest

else if Reconnect Exploratory Data then
    if ProxySink then
        if ! Received Reconnect Exploratory Data then
            Send Reconnect Reinforcement

    else if Received Reconnect Interests then
        Forward Reconnect Exploratory Data
    else
        Drop Reconnect Exploratory Data

else
    Forward Message
```

Algorithm 4 shows the logic involved in the post-gradient filter. Notice that outgoing packets require much less processing. The post-gradient LRDD filter stores the neighbor information for upstream and downstream nodes along the data path. It also drops reconnect exploratory data and reconnect reinforcement packets when appropriate.

Algorithm 4: Post-Gradient Processing for LRDD

```
if Data Message then
    Save Next Hop in Downstream Neighbor Table
    Save Previous Hop in Upstream Neighbor Table
else if Reconnect Exploratory Data then
    Drop Message
else if Reconnect Reinforcement then
    if ProxySource then
        Drop Message

else
    Forward Message
```

6.2.3 Local Flooding

LRDD makes no attempt to restrict the flooding of reconnect packets. To handle this task, we have written two additional programs. The node ID filter creates a list of neighbors from which a node has received any message. It drops flooded reconnect messages that contain a reconnect attribute with a failed node ID not in the list of known neighbors. This essentially restricts reconnect messages to the one-hop neighbors of the failed node. In a sufficiently dense network, a path back to the data flow may be found in this group of nodes. In sparse networks, ID-based local flooding may restrict flooding to such a degree that repair is impossible.

A second simple method to limit the flooding of reconnect messages is to limit the number of hops a packet may travel. The hop filter appends a hop attribute containing the maximum number of hops that the packet may travel. Packets from foreign hosts are dropped if the decremented hop count reaches zero. Otherwise, messages are forwarded with a decremented hop count. This approach works in sparse networks as long as a sufficiently large initial hop count is used. The main disadvantage of the hop-count algorithm is the difficulty in selecting an appropriate value for the maximum number of hops. In dense networks, a large hop count may result in “local” flooding which is very expensive.

6.3 Real-Time Communication Mechanism

Like the other network services, the real-time communication protocol, RTDD, was implemented in the diffusion API using attributes and filters.

6.3.1 RTDD Attributes

We defined several new attributes for RTDD. These include attributes for deadline, priority, DVM, SAT, DAT, SRT, and DAT. SVM only needs a priority value so no explicit SVM attribute is necessary. To utilize RTDD, applications simply add a deadline attribute to their publication definition. The deadline is an integer representing the deadline in milliseconds. The definition of the deadline attribute is shown below.

```
#define TIME_DEADLINE_KEY 7001
NRSimpleAttributeFactory<int> TDeadlineAttr(TIME_DEADLINE_KEY,
                                            NRAttribute::INT32_TYPE);
```

Several other attributes are necessary for the functioning of RTDD itself. We defined one attribute for each variant of RTDD. These attributes encapsulate the data that must be communicated by each version of the protocol. RTDD uses the state information in each version’s attribute to compute the priority and write it to the priority attribute. Thus, the priority attribute is used by all six versions of RTDD. These RTDD attribute definitions are shown below.

```
#define PRIORITY_KEY   7002
#define DVM_KEY        7003
#define SAT_KEY        7004
#define DAT_KEY        7005
#define SRT_KEY        7006
#define DRT_KEY        7007

/* SVM, DVM, SAT, DAT, SRT, DRT */
NRSimpleAttributeFactory<float> PriorAttr(PRIORITY_KEY,
                                          NRAttribute::FLOAT32_TYPE);
/* DVM */
NRSimpleAttributeFactory<void *> DVMAttr(DVM_KEY,
                                          NRAttribute::BLOB_TYPE);
/* SAT */
NRSimpleAttributeFactory<void *> SATAttr(SAT_KEY,
                                          NRAttribute::BLOB_TYPE);
/* DAT */
NRSimpleAttributeFactory<void *> DATtAttr(DAT_KEY,
                                          NRAttribute::BLOB_TYPE);
/* SRT */
NRSimpleAttributeFactory<void *> SRTAttr(SRT_KEY,
                                          NRAttribute::BLOB_TYPE);
/* DRT */
NRSimpleAttributeFactory<void *> DRTAttr(DRT_KEY,
                                          NRAttribute::BLOB_TYPE);
```

As the simplest case, SVM requires only one floating point number to be calculated at the source and sent to the destination – no state information is needed. The other five protocols use the priority attribute to store the calculated priority but also require another attribute to communicate state information. The DVM attribute contains the time the data packet was sent from the source. The SAT attribute holds a structure with the two timestamps used to compute end-to-end delay. The DAT attribute contains three timestamps representing the end-to-end delay timestamps and the time the data packet was sent. The SRT attribute stores the two timestamps taken by the source to compute the round trip time. Finally, DRT stores the three timestamps needed to compute the round trip time and the elapsed time. The definitions of the structures for the time-based protocols are shown below.

```
typedef struct RAP_SAT {
   EventTime ts_src_expdata;
   EventTime ts_snk_reinforcement;
}RAPSAT;

typedef struct RAP_DAT {
   EventTime ts_src_expdata;
   EventTime ts_snk_reinforcement;
   EventTime ts_src_data;
}RAPDAT;

typedef struct RAP_SRT {
   EventTime ts_src_expdata;
   EventTime ts_src_reinforcement;
}RAPSRT;

typedef struct RAP_DRT {
   EventTime ts_src_expdata;
   EventTime ts_src_reinforcement;
   EventTime ts_src_data;
}RAPDRT;
```

6.3.2 RTDD Filters

Similar to PCDD and LRDD, RTDD is implemented using a pre-gradient filter and a post-gradient filter. Each variant essentially performs the same three steps. We explain the differing details of each version in the following subsections.

1. Add RTDD information to packets at source/sink.

2. Extract RTDD information from packets at source.

3. Compute priority and append it to outgoing data packets.

SVM

In SVM, the sink adds its location information (latitude and longitude) to outgoing interest packets. The source extracts the location information and uses it along with the deadline supplied by the application to compute the priority according to Equation 3.1. This priority is stored in the priority attribute by the source and used to prioritize the data packet at each intermediate hop.

DVM

In DVM, the sink adds its location information to outgoing interest packets. The source extracts the location information and uses it along with the deadline supplied by the application to compute the priority according to Equation 3.2. The source also timestamps the packet so that intermediate nodes can compute the elapsed time. Upon receiving a data packet, intermediate nodes extract the location of the sink and the timestamp. They use this information along with their location to update the priority again using Equation 3.2.

SAT

In SAT, the source adds a timestamp to outgoing exploratory data and the sink adds a timestamp to outgoing reinforcement messages. When the source is ready to send data, it computes the difference of these two timestamps and divides it by the deadline to find the priority (Equation 4.4). The priority is stored in the priority attribute and used at each intermediate hop for queue prioritization.

DAT

In DAT, the source adds a timestamp to outgoing exploratory data and the sink adds a timestamp to outgoing reinforcement messages. The source appends a timestamp to the data packet corresponding to the time it was sent. The time difference and deadline are used to compute the initial priority, which is written to the priority attribute. Intermediate nodes subtract the elapsed time from the end-to-end delay and deadline to update the priority at each hop (Equation 4.6).

SRT

In SRT, the source stores a timestamp when exploratory data is transmitted and then saves another timestamp when the first reinforcement message is received. When the source is ready to send data, it computes the difference of these two timestamps and divides it by the deadline to find the priority (Equation 4.7). The priority is stored in the priority attribute and used at each intermediate hop for queue prioritization.

DRT

In DRT, the source stores a timestamp when exploratory data is transmitted and the saves another timestamp when the first reinforcement message is received. The source appends a timestamp to the data packet corresponding to the time it was sent. The time difference and deadline are used to compute the initial priority, which is written to the priority attribute. Intermediate nodes subtract the elapsed time from the round trip delay and deadline to update the priority at each hop (Equation 4.8).

6.3.3 Prioritized Queue

We implemented the priority queue using the timer-based approach introduced in Section 5.2.3. The implementation involved changing the typical behavior of diffusion filters that forward messages from the receive thread. Instead, we added each received message to a priority queue. The queue was implemented as a linked list of `PriorityQueueEvent` objects as defined below.

```
class PriorityQueueEvent{

public:
   Message *msg;
   handle h;
   double priority;
   PriorityQueueEvent *next;
};
```

Each data message received by the receive thread of the RTDD filter was added to the queue using the `insert()` method. The `run()` method of the RTDD filter invoked the `send()` method that served as the sending thread. It dequeued the first element in the priority queue (the highest priority message), sent it, and re-scheduled another send event in t_{timer} milliseconds. The definition of the `PriorityQueue` class is shown below. Besides the basic, insert, and dequeue operations, we also wrote several utility methods `isEmpty()`, `print()`, and `length()`.

```
class PriorityQueue {
   PriorityQueueEvent *head_;

public:
   PriorityQueue();
   void insert(Message *msg, int handle, double priority) ;
   PriorityQueueEvent * dequeue();
   bool isEmpty();
   void print();
   int length();
};
```

After the highest priority packet has been sent, the `send()` method waits for t_{timer} milliseconds before sending another packet. The value of t_{timer} determines the granularity of prioritization and affects the maximum bandwidth of the network. Larger values result in greater amounts of prioritization at the cost of throughput. Smaller values of t_{timer} allow more packets to be sent, but also reduce the amount of prioritization possible. An appropriate value should be set based on the bandwidth needed for the application.

Chapter 7

Performance Evaluation

7.1 Simulation Setup

We used ns-2.29 to simulate all three network protocols. We used the 802.11 MAC layer with directed diffusion version 3 which is supplied with ns2. Table 7.1 shows the specific settings used in our ns2 simulations. Simulation parameters unique to each protocol are explained in their respective sections.

Table 7.1: ns2 Channel Parameters

Parameters	Value
Channel	Channel/WirelessChannel
Propagation Model	Propagation/TwoRayGround
Physical Medium	Phy/WirelessPhy
MAC Layer	Mac/802_11
Queue Type	Queue/DropTail/PriQueue
Link Layer	LL
Antenna	Antenna/OmniAntenna

Our energy model follows the standard energy usage model for ns2. The parameters and values are shown in Table 7.2.

Table 7.2: ns2 Channel Parameters

Parameters	Value
Transmission Power	0.660
Reception Power	0.395
Idle Power	0.035

7.2 Clustering Mechanism (PCDD)

We evaluated our implementation of PCDD using several metrics. Since the major goal of passive clustering is to improve flooding performance, the primary metric of interest is flooding efficiency. We measured this in terms of the total number of flooded packets transmitted and the average energy consumed per node. We also measured the delivery ratio, the end-to-end delay, and the probability of disconnection for each simulation.

Our results show that PCDD significantly reduces the number of interests and exploratory data messages transmitted while also providing high delivery ratios and low end-to-end delays. The packet reduction results in a better average energy consumption, especially in dense topologies. The only disadvantage of PCDD is a slightly increased probability of disconnection. However, this probability is acceptable given the greatly improved network performance.

7.2.1 Experiment Setup

To test the performance of PCDD, we generated topologies with n nodes randomly dispersed over a 1000 m x 1000 m field. The nodes had a transmission radius of 250m. The number of nodes $n \in \{25, 50, 75, 100, 175, 250\}$. Five topologies were generated for each value of n. Each topology was used in 30 independent simulation runs of 1000 seconds. In our results, we computed the average of the 30 runs and then averaged the 5 means for the n-node topology. Hence, every data point is a result of 150 (30 * 5) runs of the simulator. Since the area was held constant and the number of nodes varies, we are effectively changing network density.

The application used for testing was a constant bit rate sender which sent one 1024 B packet every 5 seconds to the receiver for the entirety of the simulation time. We tested two cases: a scenario with one data flow and a scenario with two data flows. For the 1-flow case, we selected node 1 to be the sender and node n to be the receiver. In the 2-flow case, we also choose node 2 to be a sender and node $n-1$ as the receiver (for the second flow). Since the topologies are randomly generated, this procedure essentially creates two one-to-one data flows which are randomly located. In our results, we plot the average of Flow 1 and Flow 2 results.

We compare PCDD to the worst case flooding scenario (blind flooding), the near best case scenario (a near optimal connected dominating set), and another local knowledge-based efficient flooding protocol (probabilistic flooding). Blind flooding, performed by standard directed diffusion, represents the worst case since no effort is made to reduce redundant transmissions. To find the near best case flooding scenario, we used an evolutionary approach to find the minimal set of nodes needed for complete connectivity, a connected dominating set (CDS), for a given topology. We also compared PCDD to probabilistic flooding. Like PCDD, probabilistic flooding only utilizes information about 1-hop neighbors and operates on-line (i.e., it does not require set up ahead of time). Thus, we evaluated the performance of PCDD relative to the near best and worst possible efficient flooding algorithms as well as an equivalent efficient flooding technique.

7.2.2 Flooding Efficiency

Primarily, PCDD provides improved flooding performance by reducing the number of redundant transmissions of flooded packets. To evaluate flooding efficiency, we measured the total number of interest and exploratory data packets transmitted during the simulation. The total number of interest packets transmitted during the simulation is plotted against the number of nodes (n) for both the MAC layer (Figure 7.1) and Routing layer (Figure 7.2). While not as good as the CDS and probabilistic algorithms, PCDD provided a significant improvement over standard diffusion in every case. PCDD does not perform as well as probabilistic flooding in terms of interests because PCDD uses interests to learn the clustered structure of the network. Exploratory data receives the benefit from this "learning" phase of PCDD. Furthermore, probabilistic flooding, in this case, has an unfair advantage over PCDD in that we tuned the flooding parameter offline. Hence, we choose the lowest forwarding probability based on empirical tests run beforehand. PCDD had no such foreknowledge.

Figures 7.3 and 7.4 shows the number of interests transmitted in the 2-flow scenarios. These results follow the same trends as the 1-flow scenarios previously described. PCDD outperforms standard diffusion by about 20% while CDS and probabilistic flooding perform significantly better. This performance advantage does not carry over to exploratory data, however.

The exploratory data packets show an even more dramatic reduction with PCDD. Figures 7.5 and 7.6 depict the number of exploratory data messages transmitted at the MAC layer and Routing layer respectively for the 1-flow scenario.

The 2-flow scenarios exhibit the same behavior. PCDD soundly outperforms all the other protocols. These results are shown in Figurse 7.7 and 7.8 for MAC and Routing layers.

PCDD provides dramatically increased flooding efficiency on the order of 46% fewer flooded routing-layer packets and 86% fewer MAC-layer flooded packets over all network

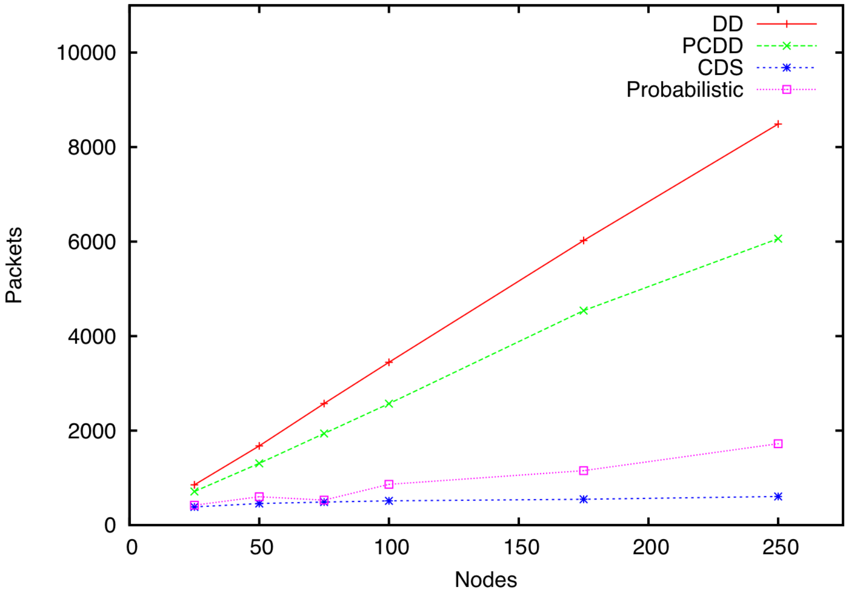

Figure 7.1: Number of MAC-layer interest messages versus number of nodes (1 flow)

sizes in the 1-flow scenarios. On average, the number of interest packets was reduced by 22% and the number of exploratory data messages was reduced by 98%. In the 2-flow cases, the number of interest packets was reduced 24% and exploratory data was reduced 99%. The total number of flooded packets was reduced 48% (MAC) and 87% (Routing). Since PCDD learns the clustered structure of the network during the interest flooding phase of route discover, it is able to restrict the flooding of exploratory data more efficiently. This allows PCDD to perform much better during the second phase of flooding (exploratory data).

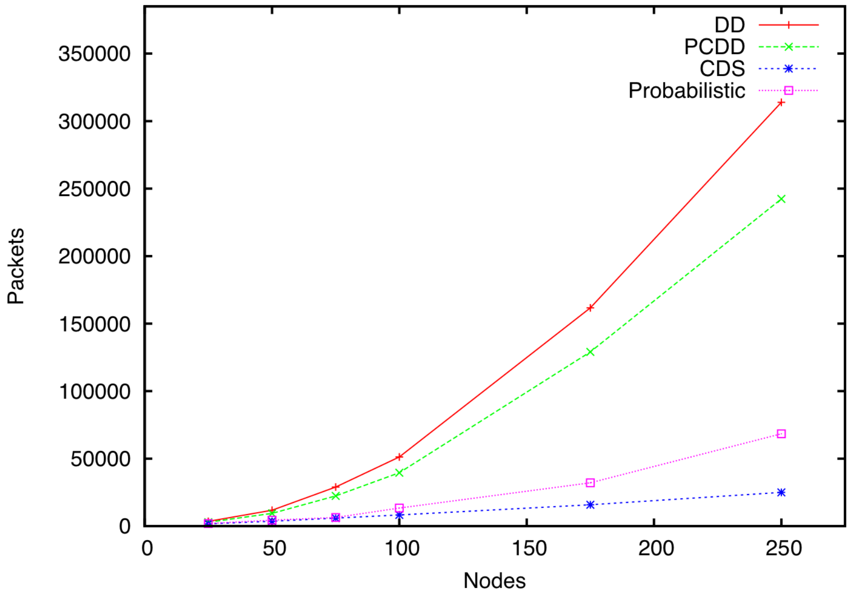

Figure 7.2: Number of Routing-layer interest messages versus number of nodes (1 flow)

This flooding reduction directly correlates to energy savings since fewer transmissions mean less power is consumed. Figure 7.9 shows the average energy consumed per node over different topology sizes for the 1-flow scenarios. Due to the large number of packets transmitted by blind flooding, standard directed diffusion performs poorly with increasingly dense networks. PCDD and CDS maintain almost constant energy consumption while probabilistic flooding has a slight increase in the larger topologies.

Once again, the 2-flow scenarios follow the same general trends as the 1-flow results. Standard diffusion performs worst, while CDS performs best, closely followed by PCDD, and probabilistic flooding.

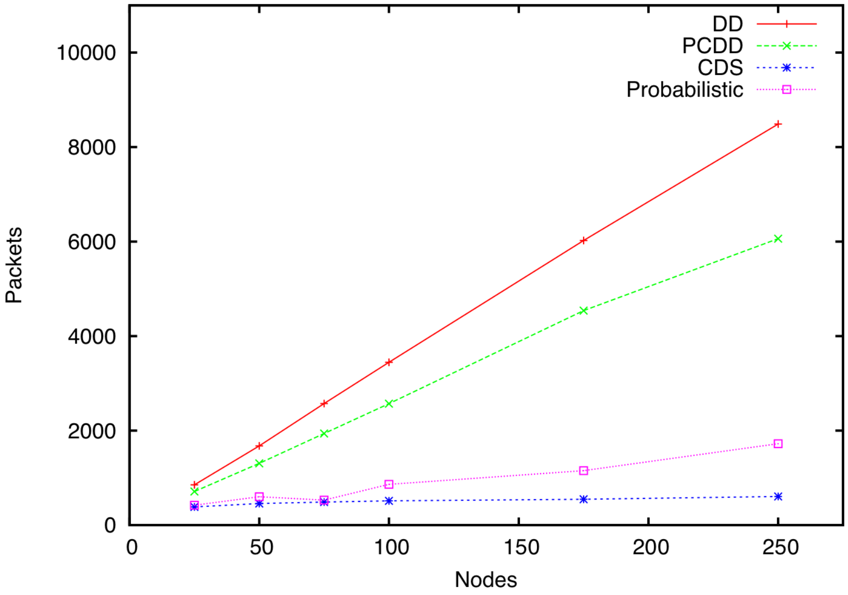

Figure 7.3: Number of MAC-layer interest messages versus number of nodes (2 flows)

7.2.3 Delivery Effectiveness

The packet delivery rates may be adversely affected by the congestion caused by flooding. Since PCDD reduces flooding-induced congestion, delivery rates can be improved when efficient flooding is performed. Thus, a secondary benefit of PCDD is increased delivery effectiveness in large, highly-connected networks. To measure delivery rates, we computed the delivery ratio, i.e., the number of packets received versus the number of packets sent. The delivery ratio for varying numbers of nodes n is shown in Figure 7.11.

Not surprisingly, the near optimal CDS produced the best delivery ratios, all better than 99%. PCDD closely followed this performance with delivery ratios greater than 98%

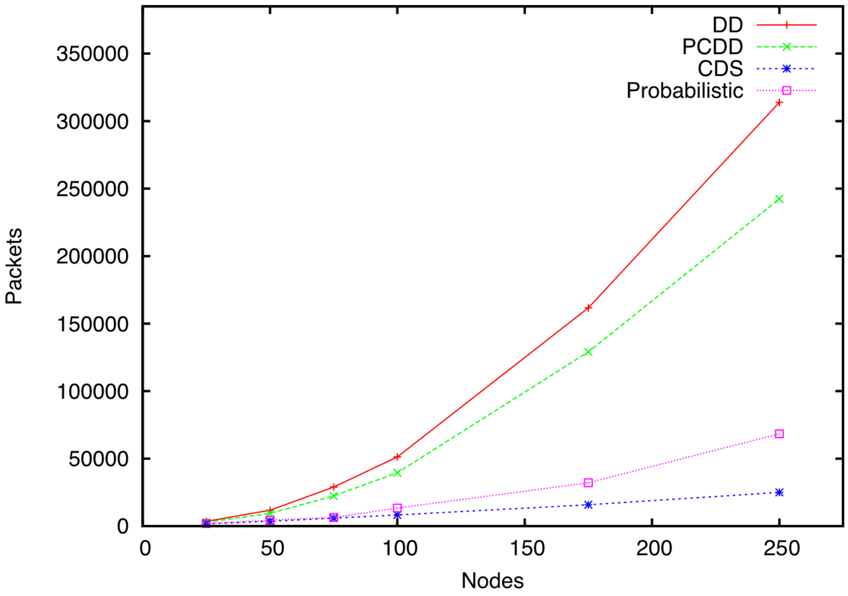

Figure 7.4: Number of Routing-layer interest messages versus number of nodes (2 flows)

even in the densest topologies. The performance of standard directed diffusion, in contrast, decreases as the number of nodes increases due to the effects of congestion. In the densest networks, diffusion only delivered 93% of the packets. Probabilistic flooding was the worst performer, however, with delivery ratios ranging from 79% to 90%.

The results of the 2-flow scenarios are shown in Figure 7.12. Like the 1-flow results, CDS performs the best, followed closely by PCDD. Directed diffusion has acceptable performance at low densities, but suffers from congestion in the large topologies. Probabilistic flooding again performs the worst with delivery ratios between 80% and 92%.

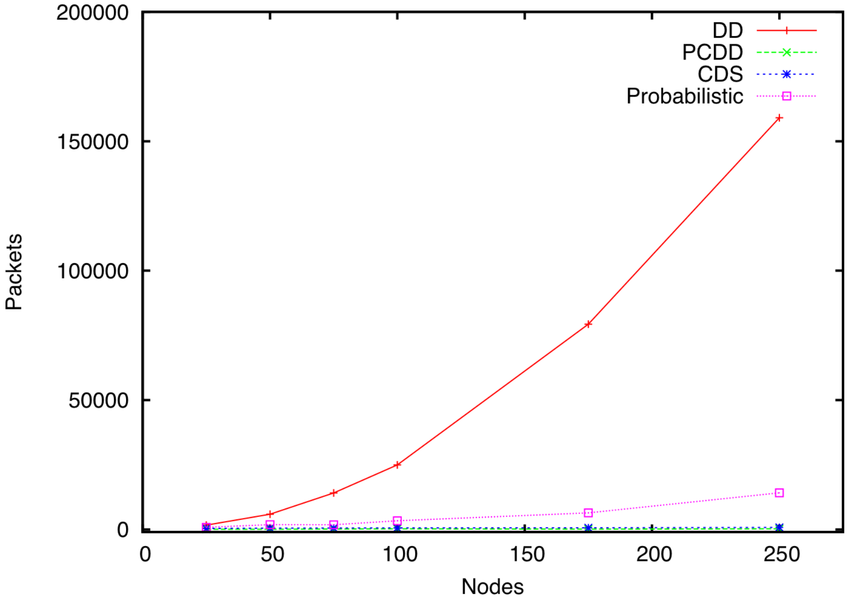

Figure 7.5: Number of MAC-layer exploratory data messages versus number of nodes (2 flows)

PCDD returned very high delivery ratios on par with optimal CDS based flooding. Thus, passive clustering provides significant advantages in terms of both flooding efficiency and delivery effectiveness. These results imply greater potential for scalability of the network and greater robustness of performance. Our results corroborate with previous work with PC over diffusion [67] in terms of improved flooding efficiency and delivery effectiveness.

7.2.4 End-to-End Delay

We also measured end-to-end delay of data packets. This metric gives another perspective on the effects of flooding-induced congestion. The average end-to-end delay of

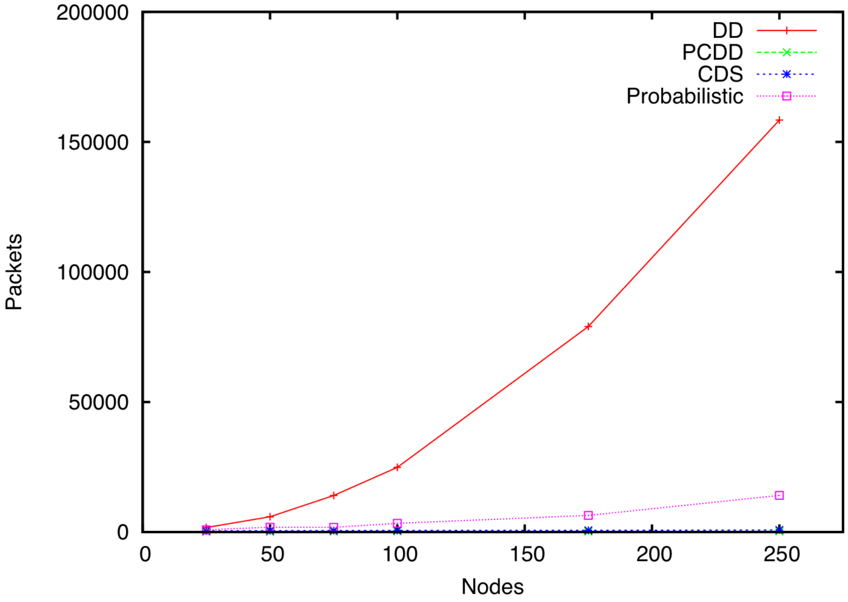

Figure 7.6: Number of Routing-layer exploratory data messages versus number of nodes (2 flows)

data packets for each set of 1-flow topologies is plotted in Figure 7.13. Notice that PCDD again closely follows the performance of the CDS algorithm. Standard diffusion suffered from incredibly high delays in the large topologies ($n = 250$) with average delays of over 500ms. Probabilistic flooding also performed poorly in terms of delay. Its average delay times increased with increasing network density.

The delays for the 2-flow scenarios (Figure 7.14) reflect similar trends as their 1-flow counterparts. CDS and PCDD are the best performers while probabilistic flooding is slightly worse. Notice that standard directed diffusion suffers from massive delays (>2000ms) in the largest topologies.

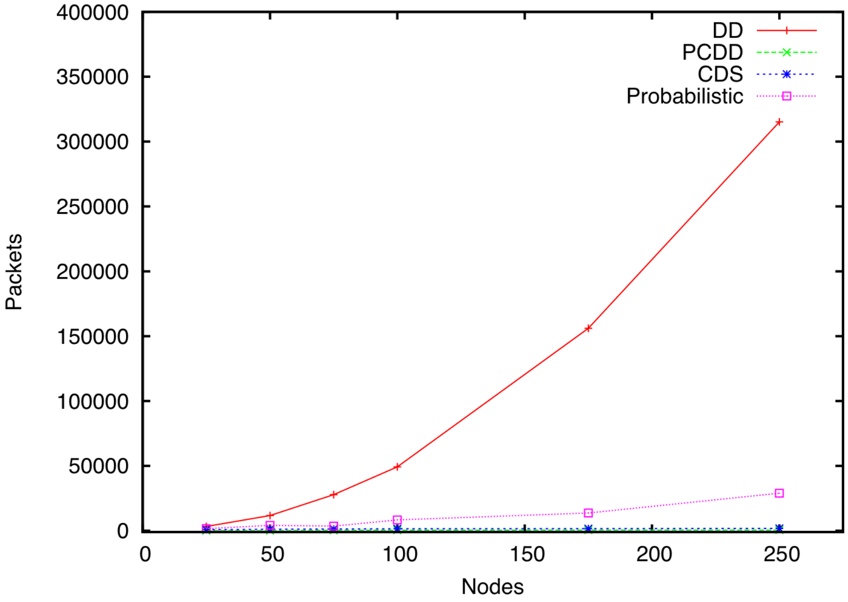

Figure 7.7: Number of MAC-layer exploratory data messages versus number of nodes (2 flows)

7.2.5 Disconnection Probability

One of the main disadvantages of PCDD is the possibility of disconnecting the network. If the gateway selection heuristic is overly aggressive, the network may be partitioned. This occurs because too many nodes are made ordinary nodes (i.e., removed from the flooding backbone). Figure 7.15 shows the average probability of disconnection for each network size. None of the other efficient flooding algorithms partitioned the network. PCDD, however, had a small probability of disconnecting the smaller topologies. For $n = 25$,

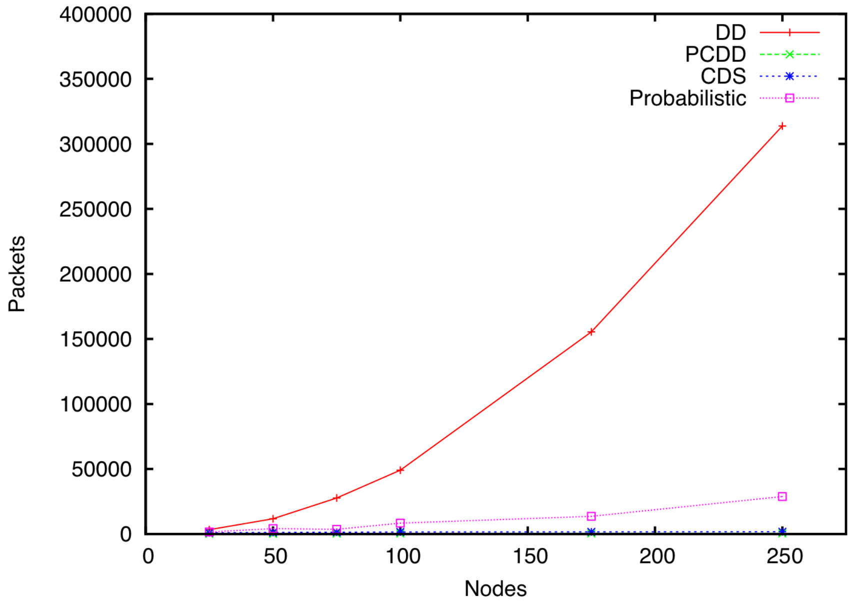

Figure 7.8: Number of Routing-layer exploratory data messages versus number of nodes (2 flows)

PCDD disconnected 8.7% of the runs. In the larger topologies, disconnection was not a problem.

In the 2-flow scenarios, disconnection presented more of a problem. As shown in Figure 7.16, PCDD suffered from small levels of disconnectivity across all topology sizes. Once again, the smaller topologies were more prone to this problem with disconnection probabilities of 8.7% in the 25-node networks. This behavior occurs because the increased number of data flows creates more "critical" nodes, i.e., nodes that cannot be ordinary without interrupting the data flow. Hence, PCDD has more opportunity to interrupt a data flow by

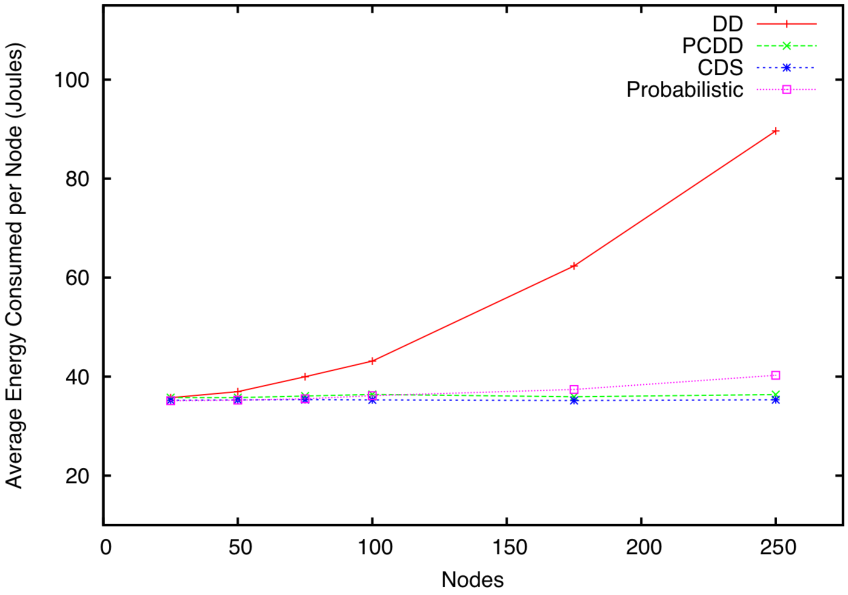

Figure 7.9: Average energy consumed per node versus number of nodes (1 flow)

wrongly identifying a critical node as ordinary. Although this behavior is disappointing, its relatively small rate of occurrence helps to mitigate the problem.

7.3 Repair Mechanism (LRDD)

We evaluated the performance of LRDD by measuring the delivery ratio for data packets and the overhead associated with local flooding. Since diffusion has no reactive repair mechanism, our addition provided significant gains in terms of delivery effectiveness. The cost of repair was generally low given the restricted flooding strategies we employed. We

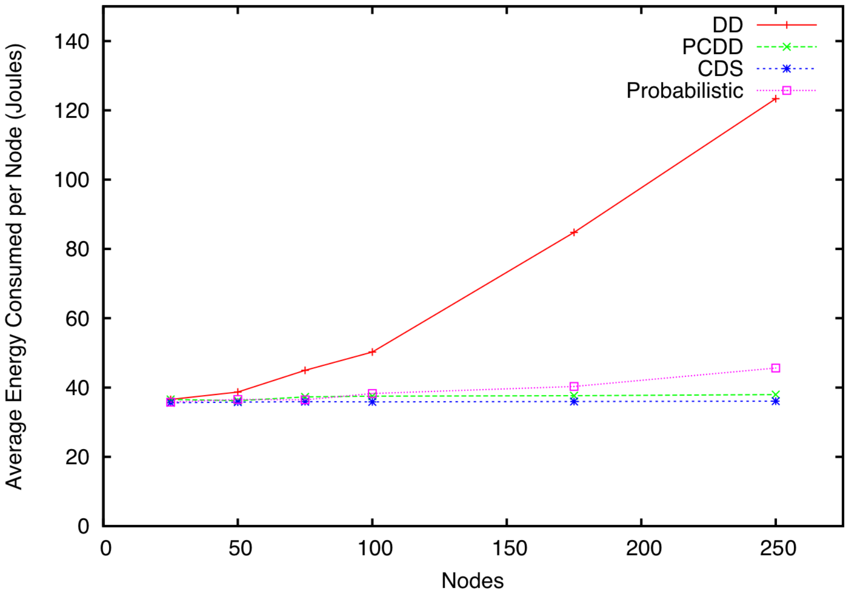

Figure 7.10: Average energy consumed per node versus number of nodes (2 flows)

explain the experimental scenario and discuss the performance of LRDD in terms of delivery effectiveness and overhead.

7.3.1 Experiment Setup

To test LRDD, we used a constant bit rate application that sent one 1024 B packet each second to one receiver application. We used five topologies with 250 nodes randomly deployed over a 2000 m x 2000 m field. To simulate the failures necessary to test LRDD, we generated failure scenarios consisting of a series of node failures over the course of the simulation. The node failures were drawn from an exponential distribution to model failures

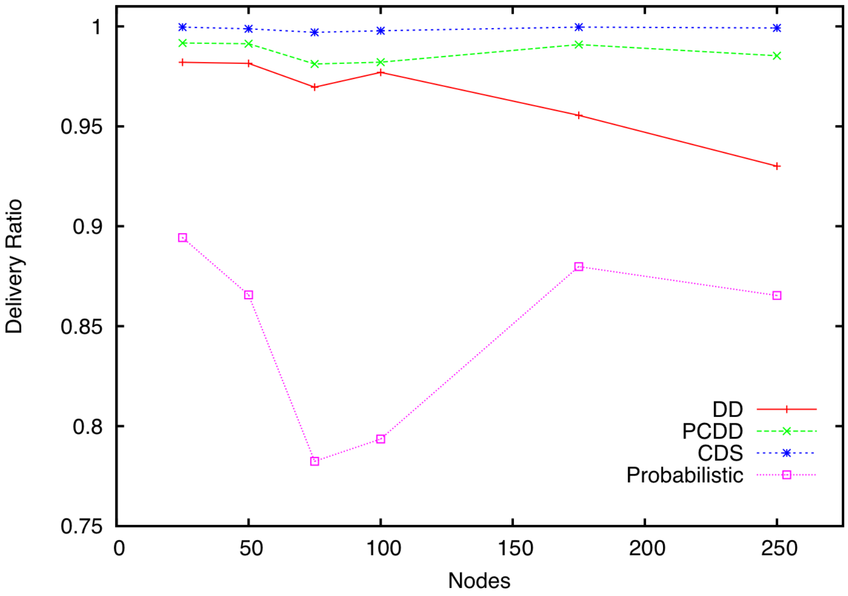

Figure 7.11: Delivery ratio versus number of nodes (2 Flows)

during the normal useful-life phase the system [69]. The mean time between failures for the exponential distribution was β where $\beta \in \{2.5,\ 5,\ 10,\ 15,\ 25\ \}$. Five failure scenarios were generated for each value of β.

Our metrics were computed over a period of 300 simulated seconds for each topology and failure scenario pair. For each of the 25 pairs, we ran five runs of the simulation with different initial seeds of the random number generator. Hence, each data point represents 125 runs of the simulation. We collected several metrics. First, we measured the number of data packets delivered to the sink. We also measured the network traffic involved in each run of the simulation. We were specifically interested in the flooded packets (interests and

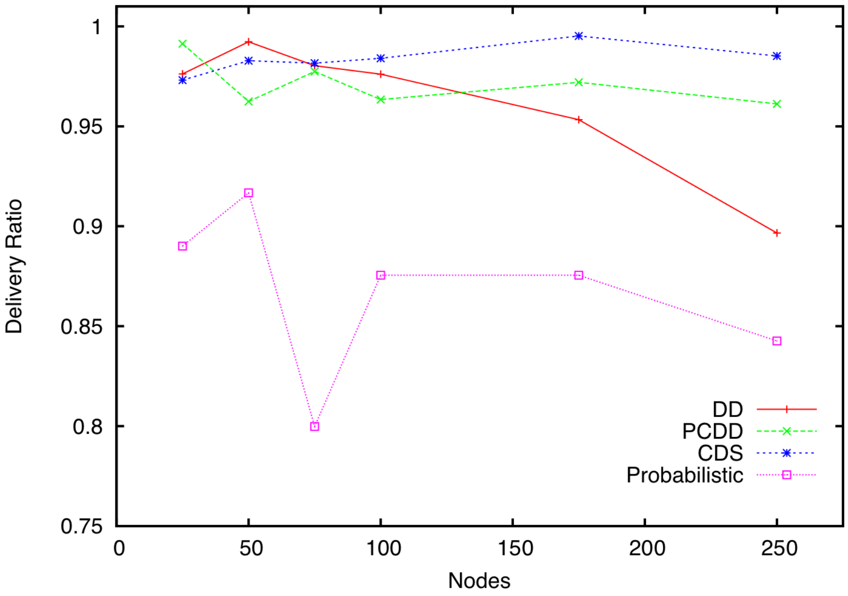

Figure 7.12: Delivery ratio versus number of nodes (2 Flows)

exploratory data) since they are most relevant to gradient repair. We calculated the average energy consumed per node. Finally, we computed two metrics to evaluate the normalized overhead involved in LRDD: flooded packets per data packet and energy per data packet.

We compare five versions of LRDD to standard directed diffusion. In LRDD with global flooding (LRDD-GF), reconnect packets were not restricted in any way and thus were flooded throughout the entire network. LRDD with ID-based local flooding (LRDD-ID) corresponds to the localized flooding strategy in which reconnect packets are only forwarded by nodes that are 1-hop neighbors of the failed node. We also tested LRDD with hop-based flooding (LRDD-Hop) with three different hop radii: 3, 4, and 5. In the figures,

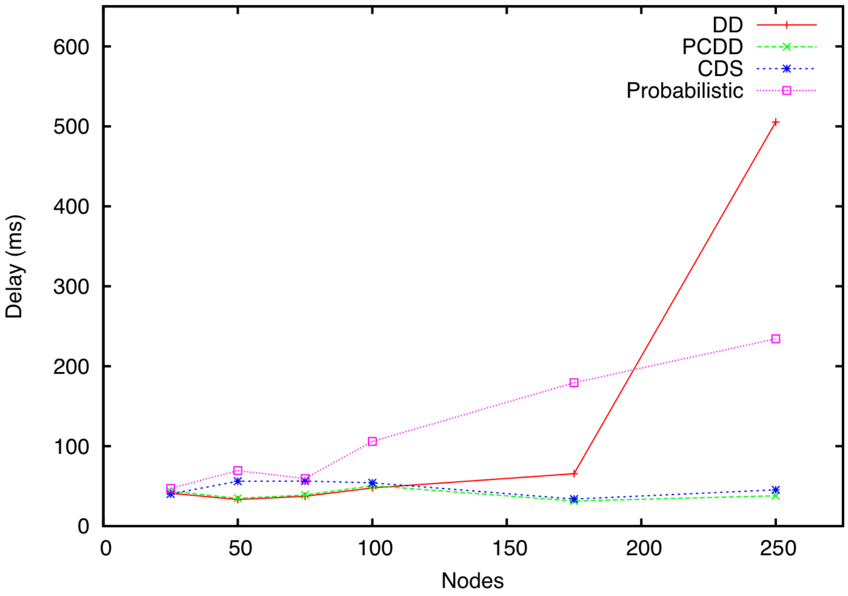

Figure 7.13: End-to-end delay versus number of nodes (1 Flow)

we denote the hop-limited protocols by appending the hop radius to the protocol name (i.e., LRDD-Hop3 means reconnect packets were forwarded in a 3-hop region).

7.3.2 Simulation Results

Packets Received

To measure LRDD's ability to recover from failures, we measured the number of data packets delivered during the course of the 300 simulated seconds. Figure 7.17 shows the average number of data packets received by the sink for the five topologies and five failure scenarios. As expected, LRDD-GF performs the best since it has the best chance of repairing

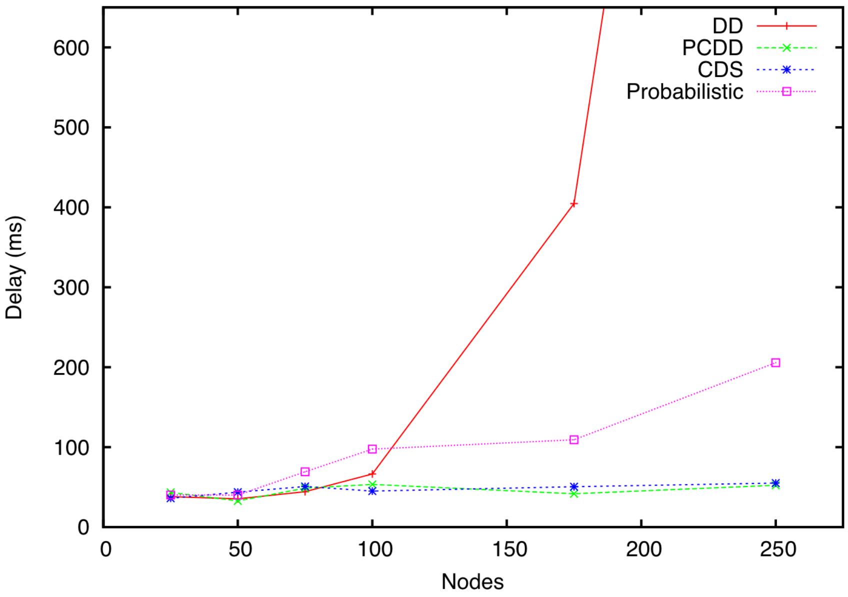

Figure 7.14: End-to-end delay versus number of nodes (2 Flows)

broken routes, albeit at a high cost (global flooding). ID-based LRDD performs second best followed closely by hop-based LRDD. ID-based LRDD performs better because it localizes the flooding to a region centered at the node which has failed. The hop-based protocols center their flooding at the proxy source and proxy sink and thus, have a lower probability of finding an alternate route around the failed node.

We have summarized the packet reception data in Table 7.3. It contains the average number of packets delivered by each protocol over all values of β. LRDD-GF performs best, followed by LRDD-ID. The hop-based protocols perform incrementally better with increasing hop size, and standard diffusion performs worst.

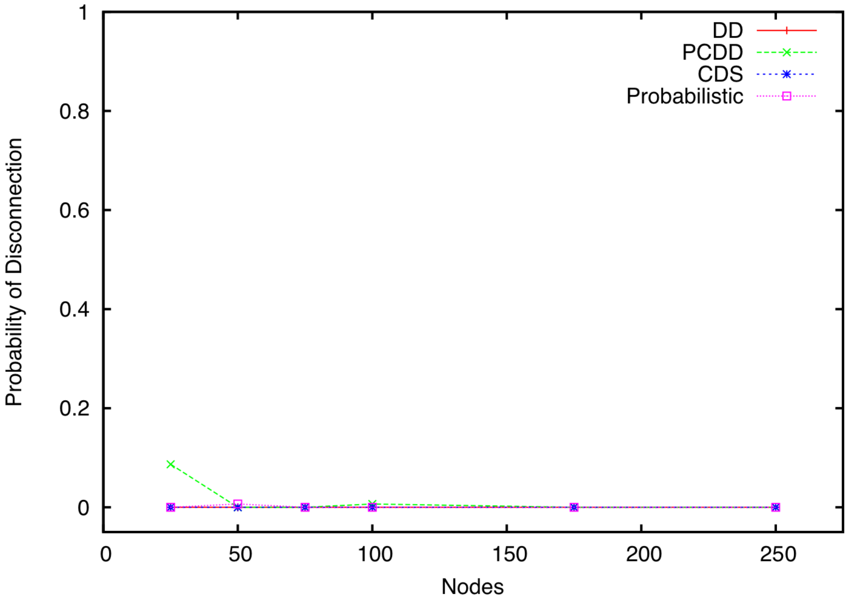

Figure 7.15: Disconnection probability versus number of nodes (1 Flow)

Flooded Packets

The next metric we evaluated was the total number of flooded packets transmitted. This metric gauges the overhead associated with the algorithms. This number includes the additional reconnect interests and exploratory data created by LRDD, thus it measures the cost incurred by reactive repair. Figure 7.18 summarizes the average number of flooded packets for each value of β. In this case, LRDD-GF is the worst performer since it produces the largest number of reconnect packets and does not attempt to restrict their transmission. The next worst performers are the hop-based local flooding algorithms in decreasing hop

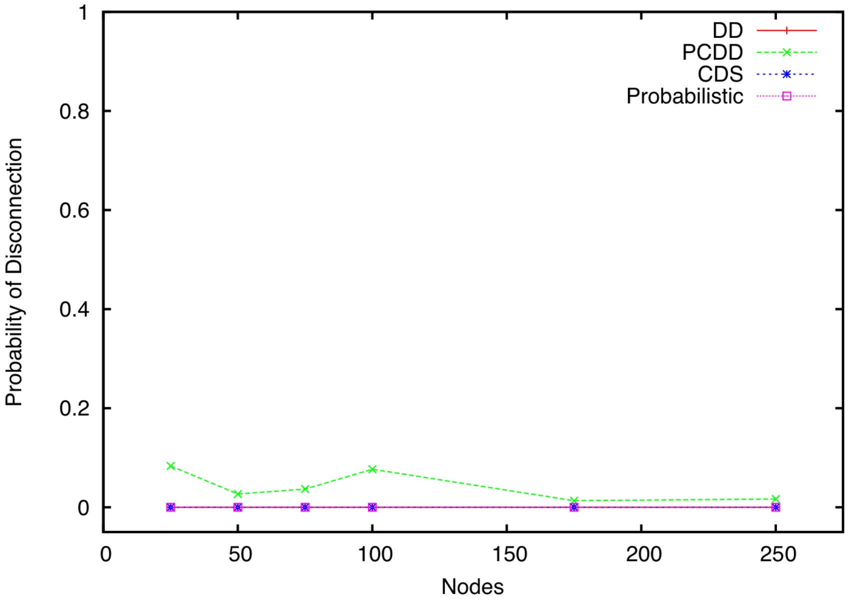

Figure 7.16: Disconnection probability versus number of nodes (2 Flows)

size. The ID-based flooding has very similar performance as the 3-hop flooding. Directed diffusion is, naturally, the best performer since it transmits no additional reconnect packets.

Table 7.4 shows the average number of flooded packets across all values of β. The same trends are apparent as in Figure 7.18. Diffusion has the lowest overhead and LRDD-GF has the highest. Notice that LRDD-ID narrowly outperforms LRDD-Hop3 in terms of the overall average.

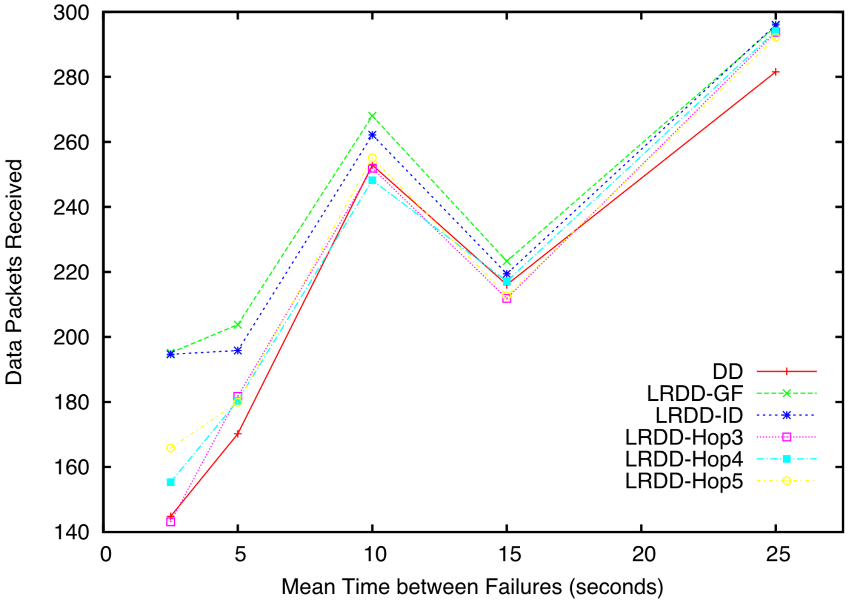

Figure 7.17: Packets delivered for each repair and flooding algorithm

Energy

Next, we evaluated LRDD in terms of energy consumption. We computed the average energy consumed per node over the simulation time. Figure 7.19 contains the results across varying values of β for each algorithm. These results necessarily correlate to the total flooded packets results discussed in the previous section. DD has the lowest energy consumption because it transmits the fewest number of packets. LRDD-GF performs worst, since it transmits the largest number of additional reconnect packets. The hop-based protocols perform slightly worse than the ID-based protocol except for a hop size of 3 where their performance is almost identical.

Table 7.3: Average packets delivered for all values of β

Repair and Flooding Algorithm	**Packets Delivered**
DD	213.10
LRDD-GF	237.18
LRDD-ID	233.62
LRDD-Hop3	216.47
LRDD-Hop4	219.03
LRDD-Hop5	221.14

Table 7.4: Average packets delivered for all values of β

Repair and Flooding Algorithm	**Flooded Packets**
DD	32088
LRDD-GF	63183
LRDD-ID	35664
LRDD-Hop3	35944
LRDD-Hop4	37682
LRDD-Hop5	39827

Table 7.5 shows the average energy consumption for each protocol over all failure scenarios. Diffusion has the lowest, followed by LRDD-Hop3 and LRDD-ID. LRDD-Hop4 and LRDD-Hop5 have slightly higher energy consumption and LRDD-GF has the highest.

Normalized Overhead

In order to gain an understanding of the trade-offs associated with the additional overhead created by LRDD, we have computed several other derived metrics using the previously discussed measurements. First, we calculate the total number of flooded packets per data packet delivered. This essentially normalizes the additional cost by the additional benefit. Figure 7.20 shows the number of flooded packets per data packet successfully delivered to the sink. LRDD-GF has the worst performance. This means that the excessive

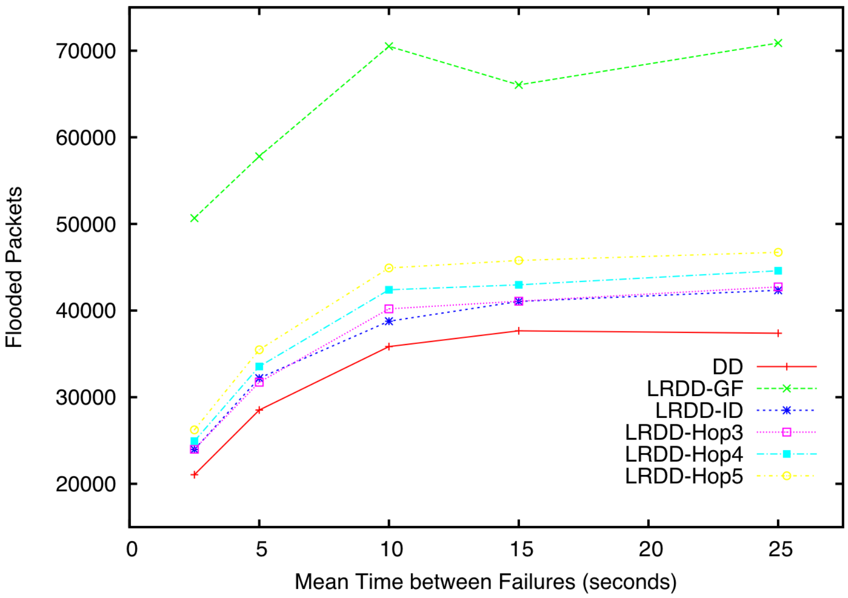

Figure 7.18: Total flooded packets for each repair and flooding algorithm

flooding costs far outweigh the additional packets delivered. The hop-based versions of LRDD perform better than LRDD-GF, but not as well as LRDD-ID and standard diffusion. LRDD-ID outperformed standard directed diffusion at the lowest values of β ($\beta = 2.5$ and $\beta = 5$), but not at the larger β values. Surprisingly, diffusion performs quite well despite its low packet delivery rate. Its strong performance is due to its low overhead. LRDD-ID had superior performance at high failure rates (low β values), but at lower failure rates, the benefit of route repair did not outweigh the cost of flooding. However, if recovery time is considered, diffusion's performance would not be strong since it only periodically repairs

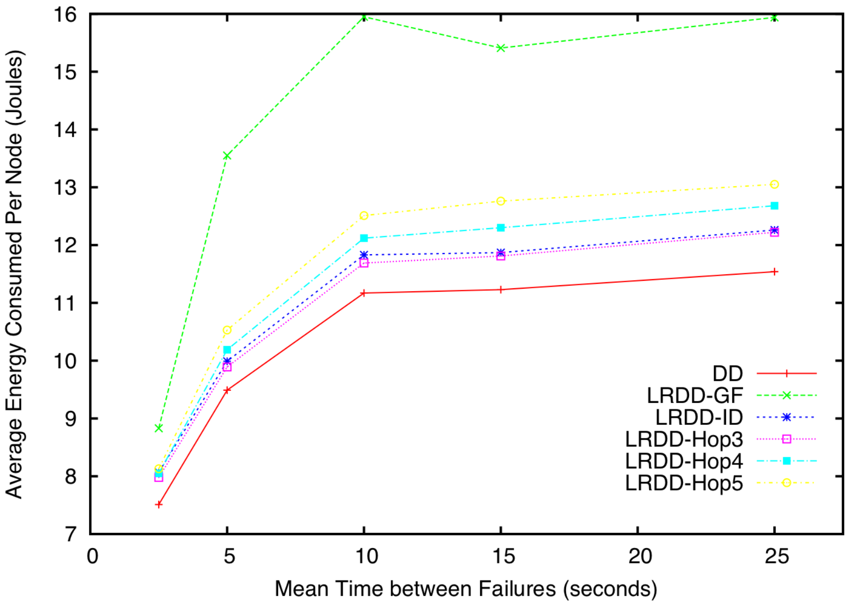

Figure 7.19: Average energy consumed per node for each repair and flooding algorithm

broken paths. On average, diffusion will take 30 seconds to repair a broken path (since each refresh cycle is 60 seconds). In contrast, LRDD recovers from repairs in 2-3 seconds.

Table 7.6 summarizes the normalized flooded packet overhead for all values of β. LRDD-ID and DD have very similar performance (0.4% difference). The hop-based versions of LRDD have slightly greater cost (11%-17% worse than directed diffusion). LRDD-GF has the worst peformance with almost twice the number of flooded packets per each data packet delivered.

Lastly, we computed the average energy consumed per node per data packet delivered. Like the previous metric, energy consumed per data packet gives an understanding of the

Table 7.5: Average packets delivered for all values of β

Repair and Flooding Algorithm	**Energy Consumed Per Node (Joules)**
DD	10.19
LRDD-GF	13.93
LRDD-ID	10.80
LRDD-Hop 3	10.72
LRDD-Hop4	11.07
LRDD-Hop5	11.40

Table 7.6: Average flooded packets per data packet for all values of β

Repair and Flooding Algorithm	**Flooded Packets Per Data Packet**
DD	45549
LRDD-GF	80247
LRDD-ID	45776
LRDD-Hop 3	50300
LRDD-Hop4	51828
LRDD-Hop5	54220

trade-offs associated with LRDD. In this case, we can evaluate the cost in terms of energy for the additional data packets delivered by LRDD. Figure 7.21 illustrates the average energy consumed per node per each data packet delivered to the sink. Generally, the worst performer was LRDD-GF since its overhead was so high. At the highest failure rate (lowest β), however, it gave the second best performance behind LRDD-ID. Overall, however, the best performers were LRDD-ID and DD. Once again, LRDD-ID bested DD at the two highest failure rates ($\beta = 2.5$ and $\beta = 5$) but not at the other rates. LRDD-Hop3 gave strong performance at all but the lowest value of β. The larger hop radii had worse performance however. To summarize, LRDD-ID had significantly better normalized energy consumption than diffusion at the higher failure rates and only slightly worse performance at

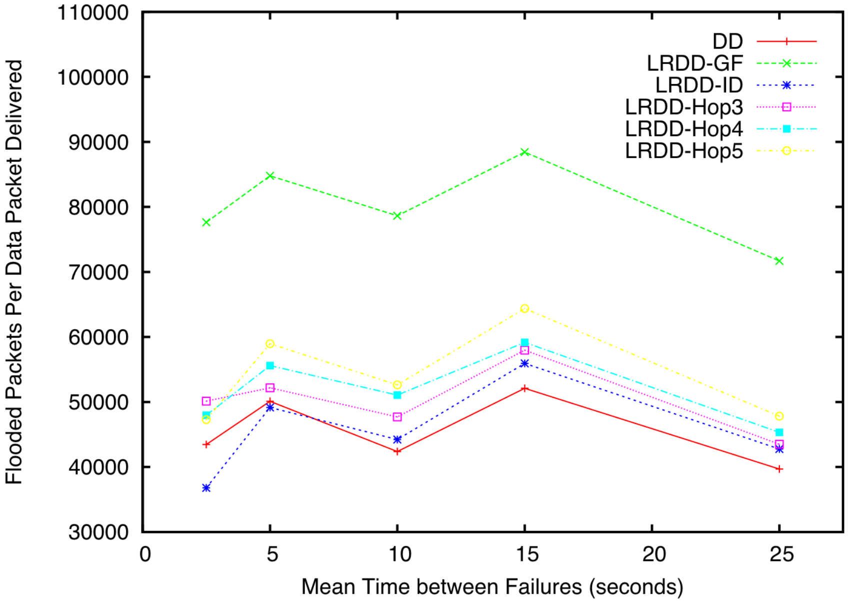

Figure 7.20: Total flooded packets per data packet delivered for each repair and flooding algorithm

the lower failure rates. LRDD-Hop3 was a close third place followed by the other hop-based protocols.

Table 7.7 contains the average energy consumed per data packet across all failure rates. LRDD-ID is the best performer followed by DD. The hop-based protocols are next best in increasing hop size. LRDD-GF is unequivocally the worst.

To summarize, LRDD significantly improves the delivery rate in the presence of node failures, however, a price must be paid for this improved behavior. We have developed several simple localized flooding techniques to reduce the cost of the protocol overhead. Our results show that the ID-based algorithm has the lowest overhead. On average, the

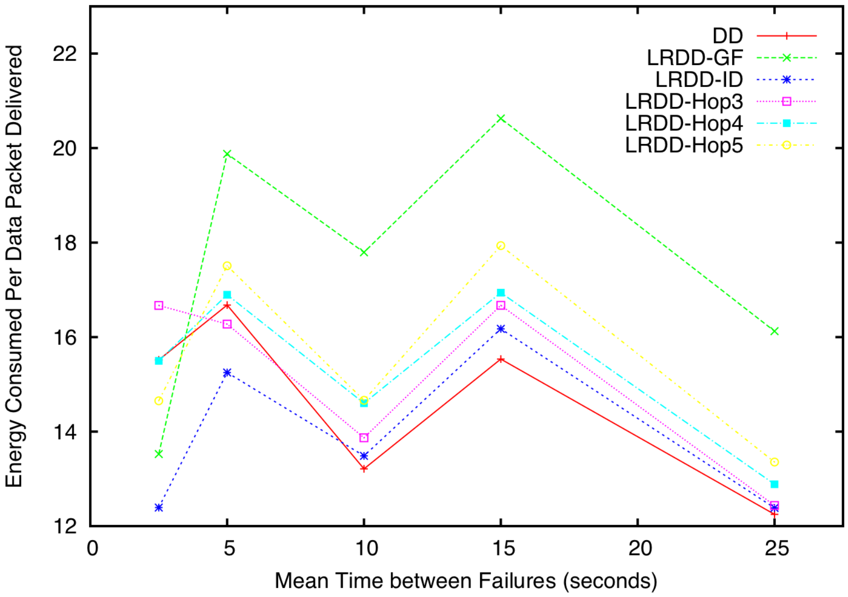

Figure 7.21: Average energy consumed per data packet delivered for each repair and flooding algorithm

ID-based localized flooding gave the best or near best performance when normalized with respect to data packets delivered. The hop-based ooding also improves packet delivery rates but at a slightly higher cost than LRDD-ID. LRDD-GF offers the best packet delivery rates but at unreasonably high cost due to global flooding. Overall, LRDD offers improved delivery rates with acceptable overhead.

Table 7.7: Average energy consumed per data packet delivered for all values of β

Repair and Flooding Algorithm	Energy Consumed Per Data Packet
DD	14.64
LRDD-GF	17.59
LRDD-ID	13.94
LRDD-Hop 3	15.18
LRDD-Hop4	15.36
LRDD-Hop5	15.62

7.4 Real-Time Communication Mechanism (RTDD)

To evaluate the effectiveness of RTDD, we measured the on-time delivery ratio of packets received at the sinks. RTDD shows significant improvement over standard diffusion in terms of timely delivery during congestion. In this section, we describe the experimental setup and compare the on-time delivery rates for directed diffusion and RTDD.

7.4.1 Simulation Setup

RTDD shows significant improvement over standard diffusion in terms of timely delivery during congestion. We tested RTDD using the bottleneck topologies shown in Figures 7.22 and 7.23. In these topologies, two or three data flows shared the same three bottleneck nodes. The bottleneck topology was chosen in order to exaggerate the effects of congestion. The metric used to evaluate RTDD was delivery ratio. Since late packets were actively dropped, packet delivery ratio corresponds to on-time delivery ratio.

The sources ran a constant bit rate application that generated one 1024 byte packet every T milliseconds where T is a constant interval parameter summed with a 20% jitter term ($T = t \pm j$). In Topology 1 $t \in \{38, 48, 54, 60, 69, 89, 125\}$, and in Topology 2 $t \in$

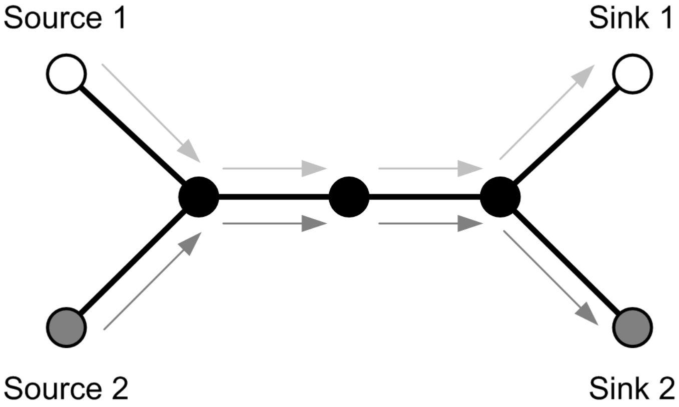

Figure 7.22: Topology 1: 2 Flows

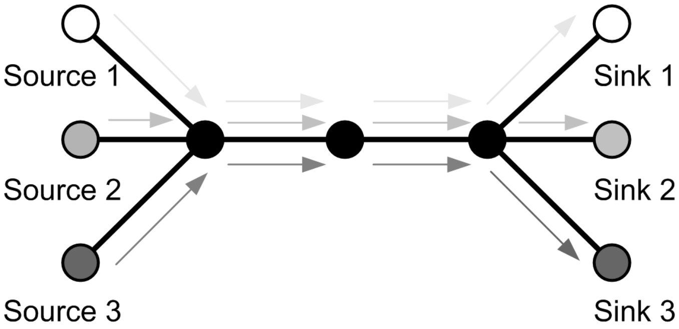

Figure 7.23: Topology 1: 3 Flows

$\{40, 45, 50, 75, 100, 125, 150, 200\}$. In both topologies $j \in [-0.2 \cdot t, 0.2 \cdot t]$. We introduced the jitter term j in order to introduce randomness in the arrival rate. The source data rates r correspond to sending one 1024 byte packet every t milliseconds on average. For the tested values of t in Topology 1, $r \in \{8.2, 11.5, 14.8, 17.1, 19.0, 21.3, 26.9\}$ KBps. For the values of t in Topology 2, $r \in \{5.1, 6.8, 8.2, 10.2, 13.7, 20.5, 22.8, 25.6\}$ KBps. Each simulation was run for 1000 simulated seconds. We ran each experiment 30 times to gain statistical confidence in the results. We report the average and the standard error of the 30 runs.

The RTDD filter queued all incoming packets and sent the highest priority packet every t_{timer} milliseconds. The t_{timer} parameter of the priority queue was set to 50 milliseconds for all the experiments. RTDD dropped packets that had missed their deadlines. This allows the network to avoid wasting bandwidth on packets which are already late. In order to make equivalent comparisons, we implemented a filter to delay standard directed diffusion packets the same amount as the RTDD filter delayed its packets. The delay filter was identical to the RTDD filter except that a priority of 0.0 was assigned to all packets, thus enforcing a FCFS ordering of the queue.

The deadlines for Flow 1 and 2 in Topology 1 were 500 ms and 625 ms respectively. For Topology 2, the deadlines were set to 500 ms, 625 ms, and 750 ms for Flows 1, 2, and 3 respectively. The lowest value (500 ms) was selected because it provided a realistic estimate of the end-to-end delay. Thus, it was possible to meet the lowest deadline in a lightly loaded network. The other deadline values were computed as 25% and 50% more than the baseline deadline.

7.4.2 Simulation Results

Since RTDD dropped packets that were late, the delivery ratio represents the percentage of packets that were delivered to the sink on-time. Packets that were not delivered were either dropped due to their lateness or lost. The overwhelming majority of the packets not delivered were actively dropped because of their lateness. Thus, the delivery ratio measures the effectiveness of the prioritization in helping packets meet their deadlines. Figures 7.24 and 7.25 show the delivery ratios of Topology 1 for Flows 1 and 2 respectively for increasing source transmission rates. The error bars represent one standard error. Notice in Figure

7.24 that at the lowest data rate (8 KBps) all seven protocols delivered over 85% of the packets on time. Past this rate, the performance of standard directed diffusion sharply dropped to almost 0%. For Flow 1, SVM and DVM maintained high delivery ratios until the transmission rates exceeded 19 KBps. Among the time-based protocols, the dynamic variants (DAT and DRT) generally performed better than their static counterparts in the intermediate region (from 11KBps to 19 KBps). At the two highest data transmission rates, none of the protocols were able to successfully deliver packets because congestion was so high.

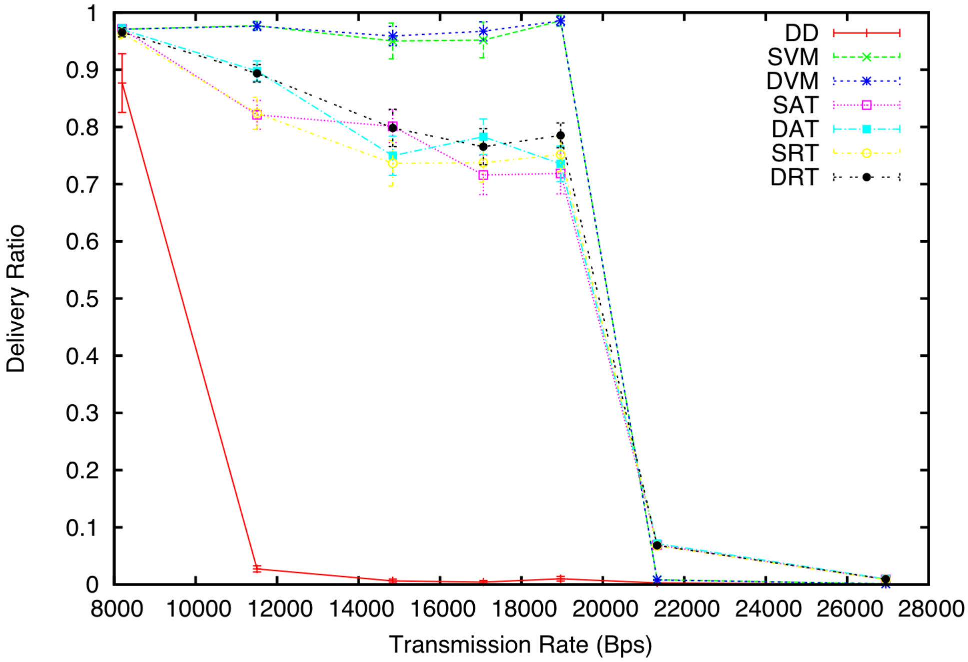

Figure 7.24: Delivery Ratio of Flow 1 (Topology 1)

Figure 7.25 shows the delivery ratio for Flow 2, the low priority flow. RTDD essentially delays packets from this flow to give preference to the high priority packets in Flow 1 (Figure 7.24). Notice that standard directed diffusion and SVM have the worst performance, closely followed by SAT and SRT. These results are expected for the static protocols since packet priorities are set once at the source. The dynamic protocols, however, update the priority of the packets at each hop. This allows initially low priority packets to be given greater preference if they are excessively penalized. From 11 KBps to 17 KBps DVM is the best performer but at the higher data rates DAT and DRT deliver more packets on time. At such high data rates, however, only about 5-10 % of the packets can be delivered successfully.

Figure 7.26 shows the average delivery ratio of both flows in Topology 1. All of the versions of RTDD provide a significant improvement over standard diffusion in the intermediate region (11 KBps to 19 KBps). The performance of the RTDD protocols falls into three classes. DVM is the best performer with a 5-15% advantage over the dynamic time-based protocols (DAT and DRT) by 5-15%. Although DVM outperforms the dynamic time-based algorithms, SVM did not outperform the static time-based protocols (SAT and SRT). Their average performance was not appreciably different (except at 19 KBps where SVM did 9% better).

Notice that the performance of the dynamic protocols gracefully degrades with increasing congestion while the static algorithms have a much sharper drop in performance. Also interesting is the performance similarity among the static protocols. SRT can perform just as well as SAT and nearly as well as SVM. Thus, we can achieve acceptable performance without location knowledge or time synchronization. The dynamic case is not quite as encouraging, however. Without location knowledge, DAT and DRT perform significantly

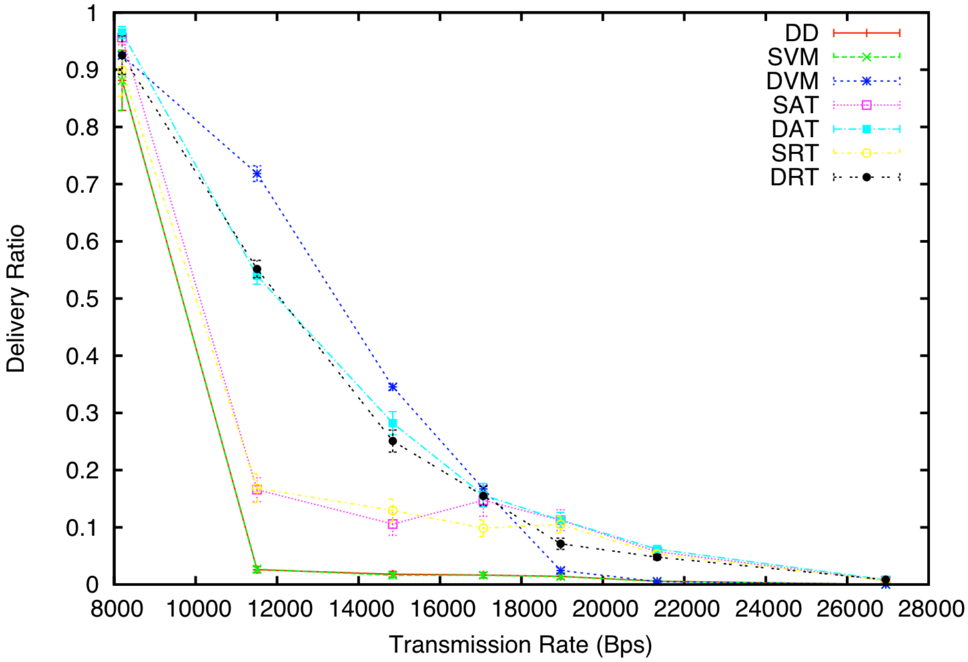

Figure 7.25: Delivery Ratio of Flow 2 (Topology 1)

worse than DVM. In summary, dynamic prioritization outperforms static, and dynamic, location-based prioritization is preferable to dynamic, time-based.

Figures 7.27 - 7.29 depict the delivery ratios for Flows 1-3 for Topology 2. As expected, the greatest performance improvement occurs in the high priority flow. As in Topology 1, RTDD significantly outperforms standard directed diffusion at all but the highest and lowest data rates (congestion levels). For Flow 1 (Figure 7.27), the location-based algorithms (SVM and DVM) outperform the time-based protocols by 10-20% in the intermediate region

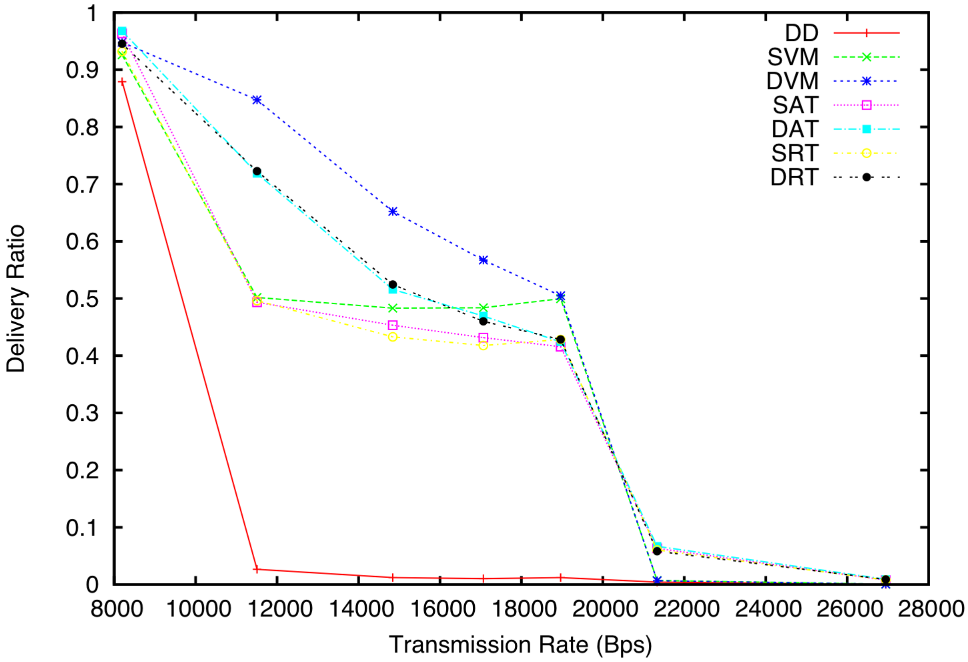

Figure 7.26: Average Delivery Ratio of Flows 1 and 2 (Topology 1)

(8 KBps to 20 KBps). The time-based protocols perform very similarly in this region, delivering 65-85% of the packets on time.

Figure 7.28 shows the performance of Flow 2, the middle priority flow. Again, diffusion has the worst performance, sharply dropping at 7 KBps. DVM is the overall best performer across all data rates. SVM has strong performance at the lower data rates (7 KBps to 13 KBps). The time-based protocols have almost identical performance from 7 KBps to 13 KBps, but DAT and DRT provide better delivery ratios than their static counterparts at the higher data rates.

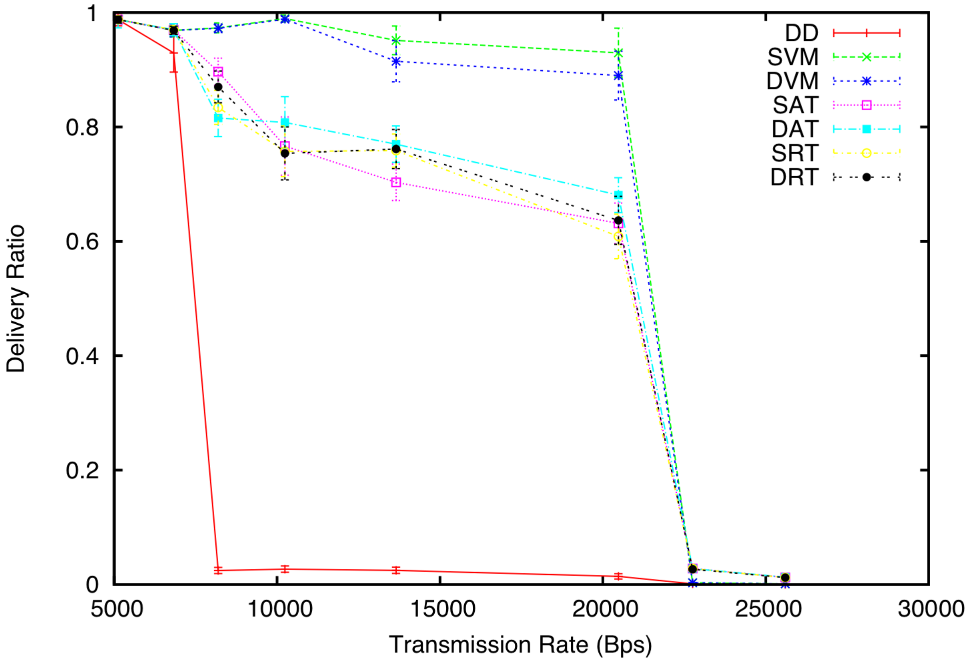

Figure 7.27: Delivery Ratio of Flow 1 (Topology 2)

Figure 7.29 summarizes the performance of the low priority flow (Flow 3). Directed diffusion and SVM had almost identical performance. DVM also had poor performance on this flow. Interestingly, the time-based versions of RTDD performed better than SVM and DVM. The dynamic time-based protocols (DAT and DRT) returned the best delivery ratios.

Figure 7.30 shows the average delivery of all three flows in Topology 2. The superior performance of RTDD versus standard diffusion is clearly shown. Among the RTDD protocols, DVM maintains a small but consistent performance advantage. SVM narrowly

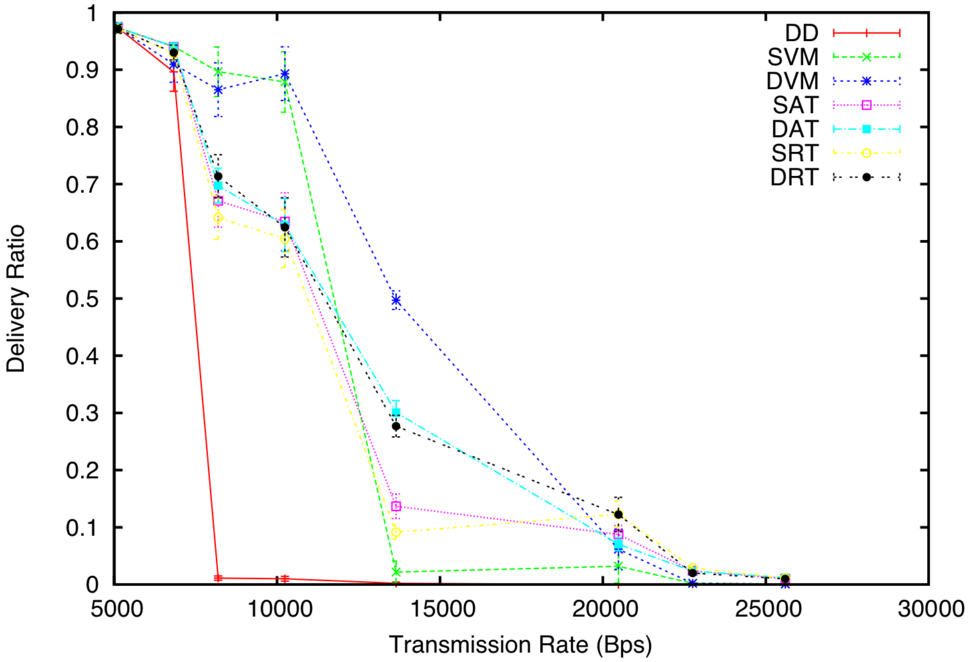

Figure 7.28: Delivery Ratio of Flow 2 (Topology 2)

edges out the SAT and SRT. As in Topology 1, SAT and SRT have no appreciable performance difference, implying that relative time differences can safely be used instead of absolute time differences. Likewise, DAT and DRT have similar performance to each other and slightly better performance than the static algorithms. Interestingly, the performance difference between the static and dynamic protocols is relatively small. This suggests that static protocols, despite their simplicity, are capable of providing good delivery rates during moderate levels of congestion. Furthermore, the small performance advantage of the location-basd protocols implies that time-based protocols may be used in networks without

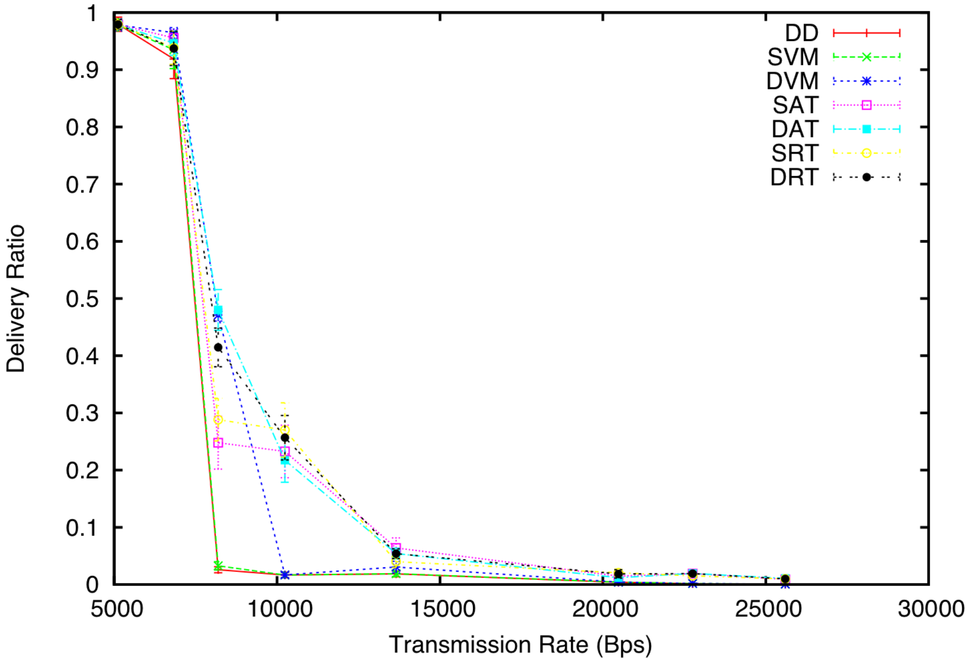

Figure 7.29: Delivery Ratio of Flow 3 (Topology 2)

localization capabilities. This significantly reduces the hardware and software requirements of the system.

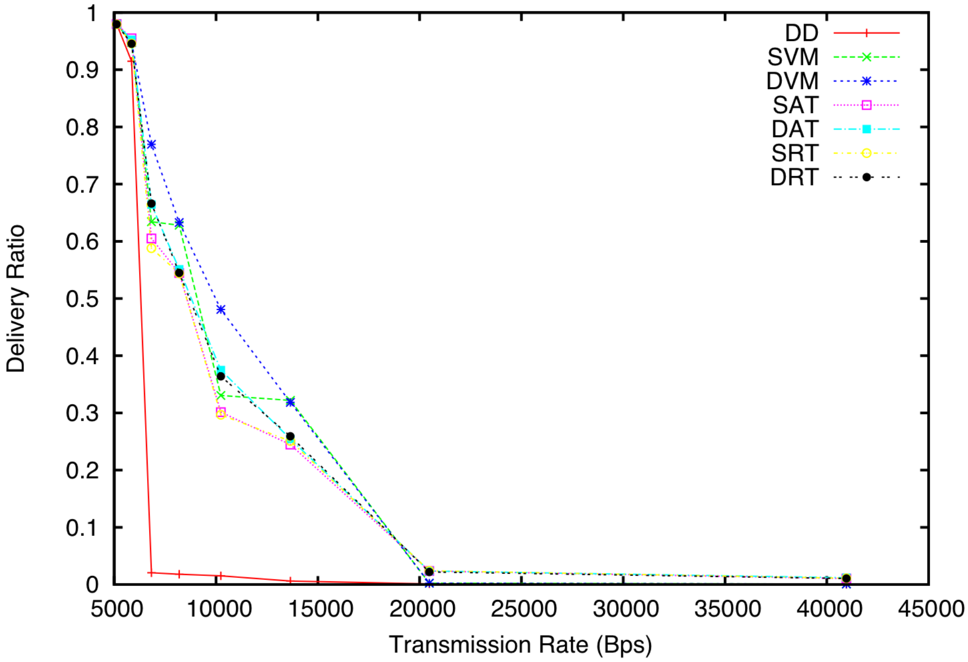

Figure 7.30: Average Delivery Ratio of Flows 1, 2, and 3 (Topology 2)

Chapter 8

Conclusions and Future Work

We have proposed, implemented, and evaluated three network services for directed diffusion. These network services improve and augment the standard directed diffusion protocol. They improve flooding efficiency, provide localized route repair, and support real-time packet delivery.

The primary contributions of our work are

- The implementation and evaluation of passive clustering for directed diffusion.
- The design, implementation, and evaluation of a reactive and localized route repair mechanism for directed diffusion.
- The design, implementation, and evaluation of a real-time communication protocol for directed diffusion that requires neither location knowledge nor time synchronization.

The three network services support the overall goals of sensor networks. Flooding efficiency and localized repair promote energy conservation by reducing unneeded transmissions. The reactive route repair mechanism improves the robustness of the network by efficiently adapting to node failure. Both PCDD and LRDD improve the scalability of the sensor network since they reduce the cost and scope of flooding. RTDD gives application developers greater control over time-critical communication, easing application development. All three network services shield high-level applications from the complexities of the underlying sensor network. By leveraging the strengths of diffusion and minimizing its weaknesses, the network services significantly improve its utility as a general purpose routing protocol.

One of the most interesting directions for future research is in considering the possible interactions among the network services. For example, LRDD could use the clusters created by PCDD to limit the flooding of repair packets. In this way, repair packets would be restricted to the clusters adjacent to the failed node. The problem with this approach is that the PC clusters may be too small to allow for successful repair.

Another possible interaction is between LRDD and RTDD. If RTDD is extended to include a congestion detection algorithm, LRDD could be used to find routes around congested regions. Repair would be performed proactively in order to improve the throughput of slow links with the same mechanism used to reactively repair node failure. The challenge of this modification is in handling the additional congestion produced by LRDD's localized flood. This mechanism will, at least temporarily, create more congestion in order to reduce congestion. Depending on the saturation of the network and the length of the new route, LRDD may not be able to improve on-time packet delivery performance. In fact, LRDD's proactive repair process may make the network congestion worse.

Finally, RTDD and PCDD could interact with each other to compose better clusters. If congestion data is maintained at each node by RTDD, then it could be used to influence the creation of clusters. A node prone to congestion might refrain from becoming a cluster head in order to reduce its workload. Although providing slight improvement, this strategy requires significant overhead in terms of state information stored by each node. Thus, the benefits do not seem to justify the cost in this instance of interaction.

Our three network services provide significant enhancements to directed diffusion in several respects. We have increased the flooding efficiency of diffusion by augmenting it with PCDD. LRDD improves the robustness of diffusion in the face of node failure without excessive flooding overhead. We have also enhanced diffusion by adding a distance and deadline-aware real-time communication protocol. By leveraging the strengths of diffusion and minimizing its weaknesses, the network services significantly improve the utility of directed diffusion as an routing protocol.

Bibliography

[1] Mark Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–10, September 1991.

[2] David Tennenhouse. Proactive computing. *Commun. ACM*, 43(5):43–50, 2000.

[3] B. Warneke, M. Last, B. Liebowitz, and K.S.J. Pister. Smart dust: communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, January 2001.

[4] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[5] Taek Jin Kwon, M. Gerla, V.K. Varma, M. Barton, and T.R. Hsing. Efficient flooding with passive clustering-an overhead-free selective forward mechanism for ad hoc/sensor networks. *Proceedings of the IEEE*, 91(8):1210–1220, Aug. 2003.

[6] Yungjung Yi, M. Gerla, and Taek Jin Kwon. Efficient flooding in ad hoc networks: a comparative performance study. In *Communications, 2003. ICC '03. IEEE International Conference on*, volume 2, pages 1059–1063vol.2, 11-15 May 2003.

[7] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 194–205, New York, NY, USA, 2002. ACM Press.

[8] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

[9] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[10] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, New York, NY, USA, 1999. ACM Press.

[11] D. Aron and Sandeep K. S. Gupta. Analytical comparison of local and end-to-end error recovery in reactive routing protocols for mobile ad hoc networks. In *MSWIM*

'00: Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, pages 69–76, New York, NY, USA, 2000. ACM Press.

[12] Nada Hashmi, Dan Myung, Mark Gaynor, and Steve Moulton. A sensor-based, web service-enabled, emergency medical response system. In *EESR '05: Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services*, pages 25–29, Berkeley, CA, USA, 2005. USENIX Association.

[13] Mitchell A. Cohen, Jakka Sairamesh, and Mao Chen. Reducing business surprises through proactive, real-time sensing and alert management. In *EESR '05: Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services*, pages 43–48, Berkeley, CA, USA, 2005. USENIX Association.

[14] S. Kapoor, K. Bhattacharya, S. Buckley, P. Chowdhary, M. Ettl, K. Katircioglu, E. Mauch, and L. Phillips. A technical framework for sense-and-respond business management. *IBM Syst. J.*, 44(1):5–24, 2005.

[15] S. Haeckel. *Adaptive Enterprise: Creating and Leading Sense-and-Respond Organizations.* Harvard Business School Press, Cambridge, MA, 1999.

[16] Michael Zink, David Westbrook, Sherief Abdallah, Bryan Horling, Vijay 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 *EESR '05: Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services*, pages 37–42, Berkeley, CA, USA, 2005. USENIX Association.

[17] C. Meinig, S.E. Stalin, A.I. Nakamura, F. Gonzelez, and H.G. Milburn. Technology developments in real-time tsunami measuring, monitoring and forecasting. In *Oceans 2005 MTS/IEEE*, Washington, D.C., September 2005.

[18] C.Meinig, S.E. Stalin, A.I. Nakamura, and H.B. Milburn. Real-time deep-ocean tsunami measuring, monitoring, and reporting system: The noaa dart ii description and disclosure. Technical report, NOAA, 2005.

[19] F.I. Gonzelez, E.N. Bernard, C. Meifg, M. Eble, H.O. Mofjeld, and S. Stalin. The nthmp tsunameter network. *National Hazards*, 35(1):25–39, 2005.

[20] Donna Casey. The thames barrier: Flood defence for london. Website, http://www.environment-agency.gov.uk /regions/thames/323150/335688/341764/, 2006.

[21] Nova: Sinking city of venice. Website, http://www.pbs.org/wgbh/ nova/venice/gates.htm, October 2002.

[22] Josh McHugh. The lost city of venice. *Wired*, 11(8), August 2003.

[23] Sammarco Paulo, Hoang H. Tran, and Chiang C. Mei. Subharmonic resonance of venice gates in waves. *Journal of Fluid Mechanics*, 349, 1997.

[24] Lee E. Harris. Combined recreational amenities and coastal erosion protection using submerged breakwaters for shoreline stabilization. Technical report, Florida Instituteof Technology, September 2005.

[25] Lee Harris. Breakwater wave attenuation.

[26] K. Mani Chandy. Sense and respond systems. In *31st Annual International Conference of the Association of System Performance Professionals*, December 2005.

[27] John S. Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*, pages 146–159, 2001.

[28] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[29] C. Perkins. Ad hoc on demand distance vector (AODV) routing. RFC 791 (Experimental), July 2003.

[30] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *Networking, IEEE/ACM Transactions on*, 11(1):2–16, Feb. 2003.

[31] Yu-Chee Tseng, Sze-Yao Ni, and En-Yu Shih. Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network. *IEEE Transactions on Computers*, 52(5):545–557, 2003.

[32] L. Orecchia, A. Panconesi, C. Petrioli, and A. Vitaletti. Localized techniques for broadcasting in wireless sensor networks. In *DIALM-POMC '04: Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 41–51, New York, NY, USA, 2004. ACM Press.

[33] Mark Ivester. Interactive and extensible runtime framework for execution and monitoring of sensor network services. Master's thesis, Auburn University, 2005.

[34] Ya Xu, John S. Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *Mobile Computing and Networking*, pages 70–84, 2001.

[35] Hyojun Lim and Chongkwon Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *MSWIM '00: Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 61–68, New York, NY, USA, 2000. ACM Press.

[36] Wei Peng and Xi-Cheng Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 129–130, Piscataway, NJ, USA, 2000. IEEE Press.

[37] Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint relaying: An efficient technique for flooding in mobile wireless networks. Technical Report Research Report RR-3898, INRIA, February 2000.

[38] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[39] C. Perkins. Multicast with minimal congestion using connected dominating sets. http://tools.ietf.org/html/draft-perkins-manet-smurf-00. IETF Internet Draft, July 2006.

[40] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Mobile Computing and Networking*, pages 85–96, 2001.

[41] J. Luna-Aceves and M. Spohn. Scalable link-state internet routing, 1998.

[42] R. G. Ogier et al. Topology dissemination based on reverse-path forwarding (TBRPF). RFC 3684 (Experimental), Feb. 2004.

[43] A. Ephremides, J.E. Wieselthier, and D.J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, 75(1):56–73, 1987.

[44] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.

[45] K. Mase, Y. Wada, N. Mori, K. Nakano, M. Sengoku, and S. Shinoda. Flooding schemes for a universal ad hoc network. In *Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE*, volume 2, pages 1129–1134, Nagoya, Japan, 2000.

[46] M. Gerla, T. Kwon, and G. Pei. On demand routing in large ad hoc wireless networks with passive clustering. In *Proceedings of the IEEE WCNC*, September 2000.

[47] Jorjeta Jetcheva David B. Johnson. Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks. In *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, pages 33 – 44. ACM, Oct 2001.

[48] D. Tian and N.D. Georganas. Energy efficient routing with guaranteed delivery in wireless sensor networks. *IEEE Wireless Communications and Networking*, 3:1923–1929, 2003.

[49] C. Gui and P. Mohapatra. A self-healing and optimizing routing technique for ad hoc networks. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, 2003.

[50] George Aggelou and Rahim Tafazolli. RDMAR: A bandwidth-efficient routing protocol for mobile ad hoc networks. In *WOWMOM*, pages 26–33, 1999.

[51] Chai-Keong Toh. Associativity-based routing for ad hoc mobile networks. *Wirel. Pers. Commun.*, 4(2):103–139, 1997.

[52] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, April 1999.

[53] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, page 1405, Washington, DC, USA, 1997. IEEE Computer Society.

[54] M. Pan, Sheng-Yan Chuang, and Sheng-De Wang. Local repair mechanisms for on-demand routing in mobile ad hoc networks. In *Dependable Computing, 2005. Proceedings. 11th Pacific Rim International Symposium on*, 2005.

[55] Genping Liu, Kai Juan Wong, Bu Sung Lee, Boon Chong Seet, Chuan Heng Foh, and Lijuan Zhu. PATCH: a novel local recovery mechanism for mobile ad-hoc networks. In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, volume 5, pages 2995–2999, October 2003.

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

[57] Tian He, J.A. Stankovic, Chenyang Lu, and T. Abdelzaher. Speed: a stateless protocol for real-time communication in sensor networks. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 46–55, 19-22 May 2003.

[58] Emad Felemban, Member-Chang-Gun Lee, and Member-Eylem Ekici. Mmspeed: Multipath multi-speed protocol for qos guarantee of reliability and timeliness in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 5(6):738–754, 2006. Student Member-Emad Felemban and Member-Chang-Gun Lee and Member-Eylem Ekici.

[59] Hyung Seok Kim, Tarek F. Abdelzaher, and Wook Hyun Kwon. Dynamic delay-constrained minimum-energy dissemination in wireless sensor networks. *Trans. on Embedded Computing Sys.*, 4(3):679–706, 2005.

[60] O. Chipara, Zhimin He, Guoliang Xing, Qin Chen, Xiaorui Wang, Chenyang Lu, J. Stankovic, and T. Abdelzaher. Real-time power-aware routing in sensor networks. *IWQoS 2006. 14th IEEE International Workshop on Quality of Service*, pages 83–92, June 2006.

[61] Y. Chen, A. Liestman, and J. Liu. Clustering algorithms for ad hoc wireless networks. *Ad Hoc and Sensor Networks*, 2004.

[62] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, New York, NY, USA, 2003. ACM Press.

[63] Hui Dai and Richard Han. Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(1):125–139, 2004.

[64] Suyoung Yoon, Chanchai Veerarittiphan, and Mihail L. Sichitiu. Tiny-sync: Tight time synchronization for wireless sensor networks. *ACM Trans. Sen. Netw.*, 3(2):8, 2007.

[65] Weilian Su and Ian F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 13(2):384–397, 2005.

[66] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 7(5):16–27, 2000.

[67] Vlado Handziski, Andreas Koepke, Holger Karl, Christian Frank, and Witold Drytkiewicz. Improving the energy efficiency of directed diffusion using passive clustering. In *EWSN 2004*, volume 2920 of *LNCS*, pages 172–187, 2004.

[68] F. Silva, J. Heidemann, and R. Govindan. Network routing application programmer's interface, USC/Information Sciences Institute, December 2002.

[69] Kishor S. Trivedi. *Probability and statistics with reliability, queuing and computer science applications.* John Wiley and Sons Ltd., Chichester, UK, UK, 2002.