Techniques for Mitigating Congestion in Wireless Sensor Networks

by

Bret Warren Hull

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[February 2005]

December 2004

Author ....
Department of Electrical Engineering and Computer Science
December 16, 2004

Certified by....... .. ..
Hari Balakrishnan
Associate Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Techniques for Mitigating Congestion in Wireless Sensor Networks

by

Bret Warren Hull

Submitted to the Department of Electrical Engineering and Computer Science on December 16, 2004, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering

Abstract

Network congestion occurs when offered traffic load exceeds available capacity at any point in a network. In wireless sensor networks, congestion causes overall channel quality to degrade and loss rates to rise, leads to buffer drops and increased delays (as in wired networks), and tends to be grossly unfair toward nodes whose data has to traverse a larger number of radio hops.

Congestion control in wired networks is usually done using end-to-end and network-layer mechanisms acting in concert. However, this approach does not solve the problem in wireless networks because concurrent radio transmissions on different "links" interact with and affect each other, and because radio channel quality shows high variability over multiple time-scales. In this thesis, we examine three techniques that span different layers of the traditional protocol stack: hop-by-hop flow control, rate limiting source traffic when transit traffic is present, and a prioritized medium access control (MAC) protocol. We implement these techniques and present experimental results from a 55-node in-building wireless sensor network. We demonstrate that the combination of these techniques can improve network efficiency by a factor of three under realistic workloads.

Thesis Supervisor: Hari Balakrishnan
Title: Associate Professor

Acknowledgments

First, I would like to thank my research advisor, Hari Balakrishnan. Through our many discussions he has provided me with invaluable insights and suggestions. The level of enthusiasm he brings to the field is only matched by his dedication to his students.

This analysis of congestion in sensor networks is joint work with Kyle Jamieson and Hari Balakrishnan. Kyle deserves a special thanks for tirelessly working with me on this project and making this research a success.

This thesis is based upon work supported by the National Science Foundation under Grant No. 0205445. This work was also supported by the MIT Project Oxygen partnership, Intel Corporation, and a Sloan Foundation Fellowship. Many thanks goes out to these partners.

I would also like to thank the many faces of the Networks and Mobile Systems group. Stan Rost deserves special thanks for helping me set up our sensor network testbed. I thank Vladimir Bychkovsky for convincing me to go on bike rides when the work got stressful. Finally, Michel Goraczko, my old officemates Nick Feamster and Michael Walfish, and the rest of NMS deserve thanks for providing help and feedback during this long process.

Finally, I am grateful to my parents who have provided me with love and support for twenty-four years and counting. Their guidance has been invaluable in getting me to where I am today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless sensor networks represent a new class of computing infrastructure. By augmenting traditional environmental sensors with computational abilities, researchers have enabled a new form of intelligent environmental monitoring. This ability to do fine-grained, autonomous sensing has created a new set of system design challenges. Many emerging sensor networks operate collectively and are composed of small, wireless, resource-constrained computers that are designed to analyze and actuate the environment. More importantly, it is the very fact these networks work in a collaborative fashion under harsh conditions that presents many unique challenges when designing network protocols. This thesis analyzes one of these challenges: *congestion control.*

1.1 Wireless sensor networks

Sensor computing systems are designed to cope with a distinct set of challenges that make them unlike traditional computer systems. These challenges arise from the fact that sensors are often embedded in the physical world and must operate autonomously. To illustrate more clearly what defines a sensor network and how such a network differs from other systems, we present a brief discussion of some of the properties that many sensor networks exhibit.

1. *Highly distributed:* Sensor networks are composed of a large number of individual nodes that cooperate to perform a collective task, such as event detection [45], collaborative filtering [57], or data aggregation [32]. Scalability is one of the key requirements for any sensor system and designers often target systems with hundreds or even thousands of nodes.

2. *Small form-factor:* Nodes must be small enough to unobtrusively monitor the environment in which they are placed. Today's sensors (e.g., the Crossbow *Mica2* described in Chapter 2) are about the size of 2 AA batteries.

3. *Wireless:* Nodes communicate by forming ad-hoc wireless networks. These networks consist of dynamic, multi-hop routes, allowing nodes in distant regions of the sensor network to communicate with each other as well as with access points.

4. *Situated:* Sensor networks interface with the physical world. Depending on the purpose of the network, nodes may be equipped with sensors that detect light, temperature, sound, acceleration, etc.

5. *Unattended:* Sensor networks are expected to operate in remote environments with little human interaction for months or years at a time. Physically interacting with the nodes to reboot them, upgrade software, or change batteries often is not feasible.

6. *Low-power:* Sensor nodes are often battery or solar powered and need to operate continuously for months or years. Due to the current limitations of energy technologies, sensor network hardware and software must be designed to minimize energy consumption.

7. *Failure tolerant:* The environments in which sensors operate are often harsh. Such factors as humidity, changes in temperature, or human tampering all contribute to node failure. Sensor systems must be designed with the expectation that some nodes will fail.

These seven properties provide the intuition for why designing sensor systems is difficult. Although much research in sensor networking takes many of these design issues in new directions, the principal challenge when building sensor systems revolves around *simultaneously* handling the constraints previously listed. Many of the protocols and architectures used in today's network systems are not adequate. In this thesis we will focus on one of these areas, congestion control, and will develop and analyze several algorithms tailored to wireless sensor networks.

1.2 Wireless sensor networks in practice

Researchers have deployed wireless sensor networks for a variety of applications. Currently, one of the most prevalent classes of applications is environmental monitoring. Figure 1-1 depicts the functional components of a hypothetical sensor network deployed to monitor a wildlife habitat. Small, battery-powered, radio-equipped sensor nodes are placed throughout the environment to monitor such phenomena as precipitation, temperature, sunlight exposure, or bird nest occupancy. Because these low-power nodes are dispersed over a large area, each node must act as a router, helping forward data through a multi-hop network. The sensor readings will eventually reach an access point, where they can be logged and analyzed by researchers. If researchers need to interact with the network, these nodes might run a distributed query processing system called TinyDB [32]. Such a system allows users to interact with a monitoring network by issuing simple queries that run in real-time on the network.

One of the earliest large-scale deployments of wireless micro-sensors took place on Great Duck Island in Maine [47]. Researchers deployed 150 devices in a remote wilderness to monitor the environmental and nesting conditions of a small sea bird called a Leach's Storm Petrel. The system utilized a tiered architecture to deliver data from the sensors in the field to the biologists in the lab. The first tier of the network consisted of Mica2Dots (described in detail in Chapter 2) that recorded light intensity, temperature, passive infrared, and humidity. These values were forwarded over a

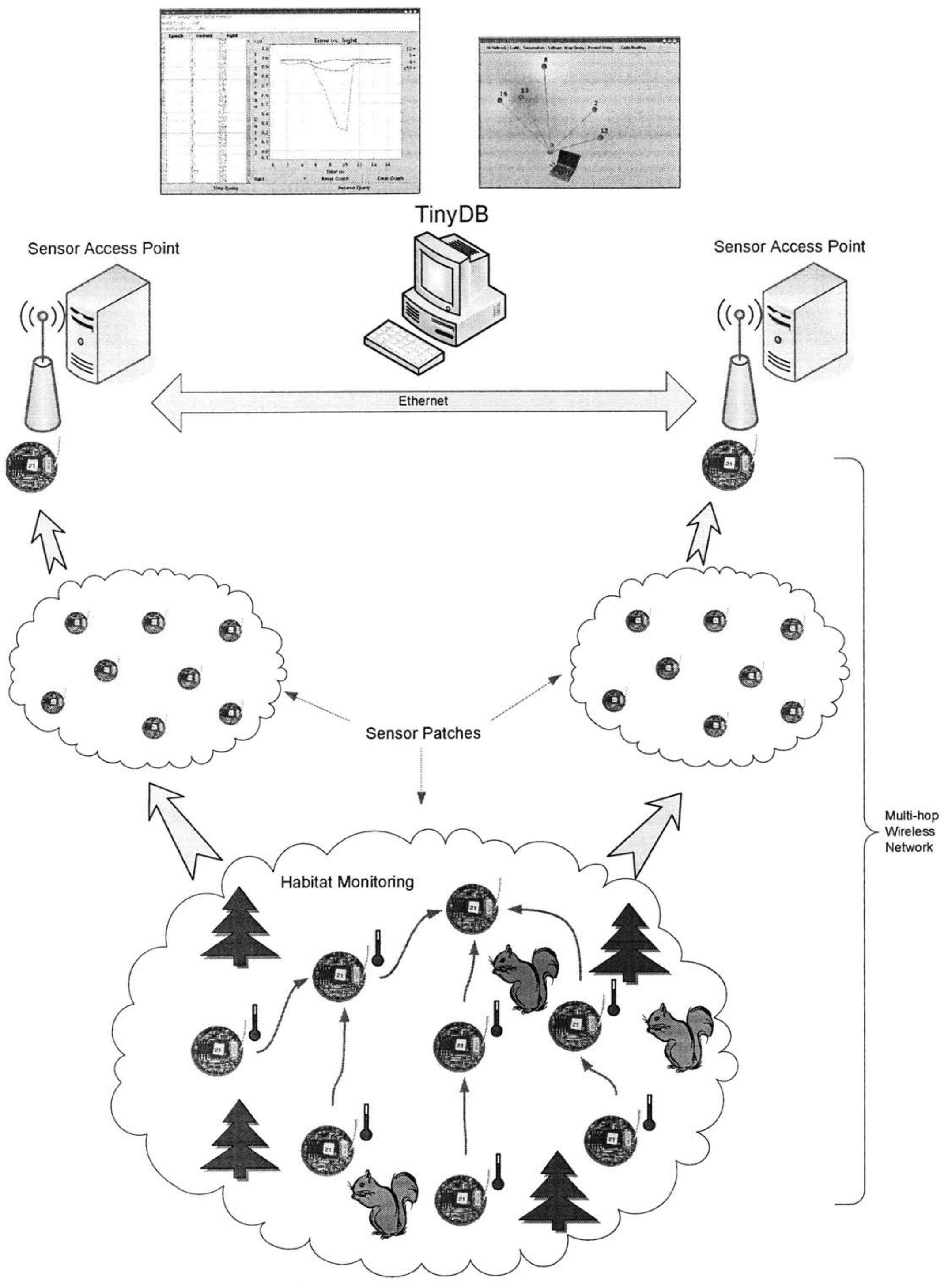

Figure 1-1: Pictured above is a schematic view of a hypothetical wireless sensor network deployed for habitat monitoring. Patches of sensor nodes (*bottom*) monitor environmental conditions in the wildlife habitat. These nodes, along with other nodes from different regions of the habitat (*center*), transmit their readings using a multi-hop wireless network. Once the sensor data reach reaches a sensor access point, the values are stored in a database (*top*). Users can monitor this data in real-time or look at historical trends.

multi-hop network to one of many base stations outfitted with high-gain antennas and car batteries. The base stations communicated over long-range wireless links to a laptop in the island's lighthouse. From this laptop, readings would be sent to a remote database server using a satellite connection. The readings could then be analyzed by biologists studying the nesting and breeding patterns of the birds.

1.3 Congestion in wireless sensor networks

Provisioning a wireless sensor network so that congestion is a rare event is extremely difficult. Sensor networks deliver myriad types of traffic, from simple periodic reports to unpredictable bursts of messages triggered by external events that are being sensed. Even under a known, periodic traffic pattern and a simple network topology, congestion occurs in wireless sensor networks because radio channels vary in time (often dramatically) and concurrent data transmissions over different radio "links" interact with each other, causing channel quality to depend not just on noise, but also on traffic densities. Moreover, the addition or removal of sensors, or a change in the report rate can cause previously uncongested parts of the network to become under-provisioned and congested. Finally, when sensed events cause bursts of messages, congestion becomes even more likely.

In traditional wired networks and cellular wireless networks, buffer drops and increased delays are the symptoms of congestion. Over the past many years, researchers have developed a combination of end-to-end rate (window) adaptation and network-layer dropping or signaling techniques to ensure that such networks can operate without collapsing from congestion. In addition to buffer overflows, a key symptom of congestion in wireless sensor networks is a degradation in the quality of the radio channel caused by an increase in the amount of traffic being sent in *other* parts of the network. Because radio "links" are not shielded from each other in the same way that wires or provisioned cellular wireless links are, traffic traversing any given part of the network has a deleterious impact on channel quality and loss rates in other parts of the network. Poor and time-varying channel quality, asymmetric communication channels, and hidden terminals all make even well-regulated traffic hard to deliver. In addition, under traffic load, multi-hop wireless sensor networks tend to severely penalize packets that traverse a larger number of radio hops, leading to large degrees of unfairness.

While higher level sensor network communication protocols, such as Directed Diffusion [21], PSFQ [51], or RMST [46], can be adapted to limit the amount of congestion generated, there remains a common need for congestion control mechanisms in each. This thesis posits that congestion control algorithms should be abstracted away from the applications and data dissemination protocols and moved into the network layer of wireless sensor networks.

1.4 Contributions

This thesis studies three congestion control techniques that operate at different layers of the traditional protocol stack, and shows that the adverse effects of network congestion can be greatly alleviated when they operate in concert. The first technique is *hop-by-hop flow control*, in which nodes signal local congestion to each other via backpressure, reducing packet loss rates and preventing the wasteful transmissions of packets that are only destined to be dropped at the downstream node. The second technique is a *source rate limiting* scheme to alleviate the serious unfairness toward sources that have to traverse a larger number of wireless hops. The third technique is a *prioritized MAC* layer that gives a backlogged node priority over non-backlogged nodes for access to the shared medium, thus avoiding buffer drops. We combine these techniques into a strategy called *Fusion*. In isolation, each technique helps somewhat, but when acting in concert, Fusion dramatically improves network efficiency, fairness, and channel loss rates. These experimental findings, together with the design details of the aforementioned mechanisms, are the primary contributions of this thesis.

In developing the techniques, we borrow heavily from previous work. For example, our hop-by-hop flow control scheme was inspired by work done on wired networks and by recent work [52] that applies the idea to wireless sensor networks. Our source rate limiting scheme bears some similarity to, and was inspired by, previous work as well [54]. However, the details of our schemes and their synergistic operation, as well as a detailed experimental evaluation of these approaches both individually and in concert, are important novel contributions of this thesis. We evaluate each scheme in a 55-node indoor wireless sensor network testbed. In our testbed, Fusion improves efficiency by a factor of three and fairness by a factor of more than two.

In the next chapter we present background information about sensor network hardware and software, as well as our testbed setup. In Chapter 3 we make the case for why congestion is undesirable in sensor networks and propose several metrics for measuring network performance. In Chapter 4 we present several techniques aimed at mitigating congestion in these networks. In Chapter 5 we study each congestion control mechanism in isolation and in concert, over different traffic patterns. Finally, in Chapter 6 we summarize our main results and describe future directions.

Chapter 2

Background

Before we present the details of congestion (Chapter 3) or how to mitigate its effects (Chapter 4), we need to understand the types of hardware and software that compose a sensor network and how these networks operate. In Section 2.1 we survey several hardware platforms to provide context for our later discussion. Just as important as the hardware is the software; in Section 2.2 we present an overview of one popular programming environment for sensor networks, TinyOS. Since many of the results presented in the later chapters are from a deployed sensor network testbed, we describe the specifics of this testbed in Section 2.3. Additionally, in that section we describe the details of radio communication and medium access control in wireless sensor networks. Finally, since much of this thesis builds on work from a number of areas, we close this chapter by presenting related work in Section 2.4.

2.1 Sensor network platforms

As with any hardware class, the state-of-the-art platform in sensor networks is a moving target. This section presents a brief overview of several hardware platforms that are used to build wireless sensor networks. We focus on micro-sensors available to the academic research community. Note that we sometimes use the term *mote* throughout the rest of this thesis to refer to an individual micro-sensor node. Table 2.1 and Figure 2-1 provide side-by-side comparisons of the platforms presented in this

	Mica	Mica2	MicaZ	Telos	iMote	Stargate
CPU	Atmel ATmega 103L	Atmel ATmega 128L		TI MSP430	Intel ARM7	Intel PXA-255 XScale
Type	4 MHz, 8 bit	7.37 MHz, 8 bit		8 MHz, 16 bit	12 MHz, 32 bit	400 MHz, 32 bit
RAM	4 KB			2 KB	64 KB	64 MB
ROM	128 KB			60 KB	512 KB	32 MB
Radio	TR100	CC100	CC2420		Bluetooth	various
Radio specs	40 Kbps, 433/916 MHz	38.4Kbps, 315/433/ 915 MHz	250 Kbps, 2.4 GHz		720 Kbps, 2.4 GHz	varies
Interface	51-pin Mote connector, serial			12-pin header, USB	serial, USB, I2C	serial, CF, PCMCIA
OS	TinyOS					Embedded Linux

Table 2.1: A side-by-side feature comparison of six sensor network platforms.

section. It is important to remember that sensor systems are not a recent development; the military and industry have been using sensor systems in one form or another for decades. What separates today's sensors from those of the past is the emphasis on large-scale, energy-constrained, wireless networks.

2.1.1 Crossbow Mica family

The Mica family of wireless sensor nodes was originally developed at U.C. Berkeley as a research platform. The technology has since been commercialized by Crossbow Technology Inc. The Mica family of motes separates sensing and computation into distinct components. Each node consists of a processing board, which contains the radio and microcontroller, and one or more data acquisition boards, which may contain one or more sensors (i.e., photocell, thermistor, microphone). The sensor board connects to the processing board via a double-sided 51-pin connector, which allows the sensor boards to be stacked. This connector has since become a de facto standard and is included in many other platforms for compatibility.

The first generation Mica was built around a 4 MHz, 8 bit Atmel ATmega microcontroller [9]. This low-power RISC processor has 4 KBytes of RAM and 128 KBytes of read-only flash used to store the program binary. For communication, the Mica uses a RFM TR1000 [43] radio operating in the 916 MHz band at 40 Kbits/s. The device

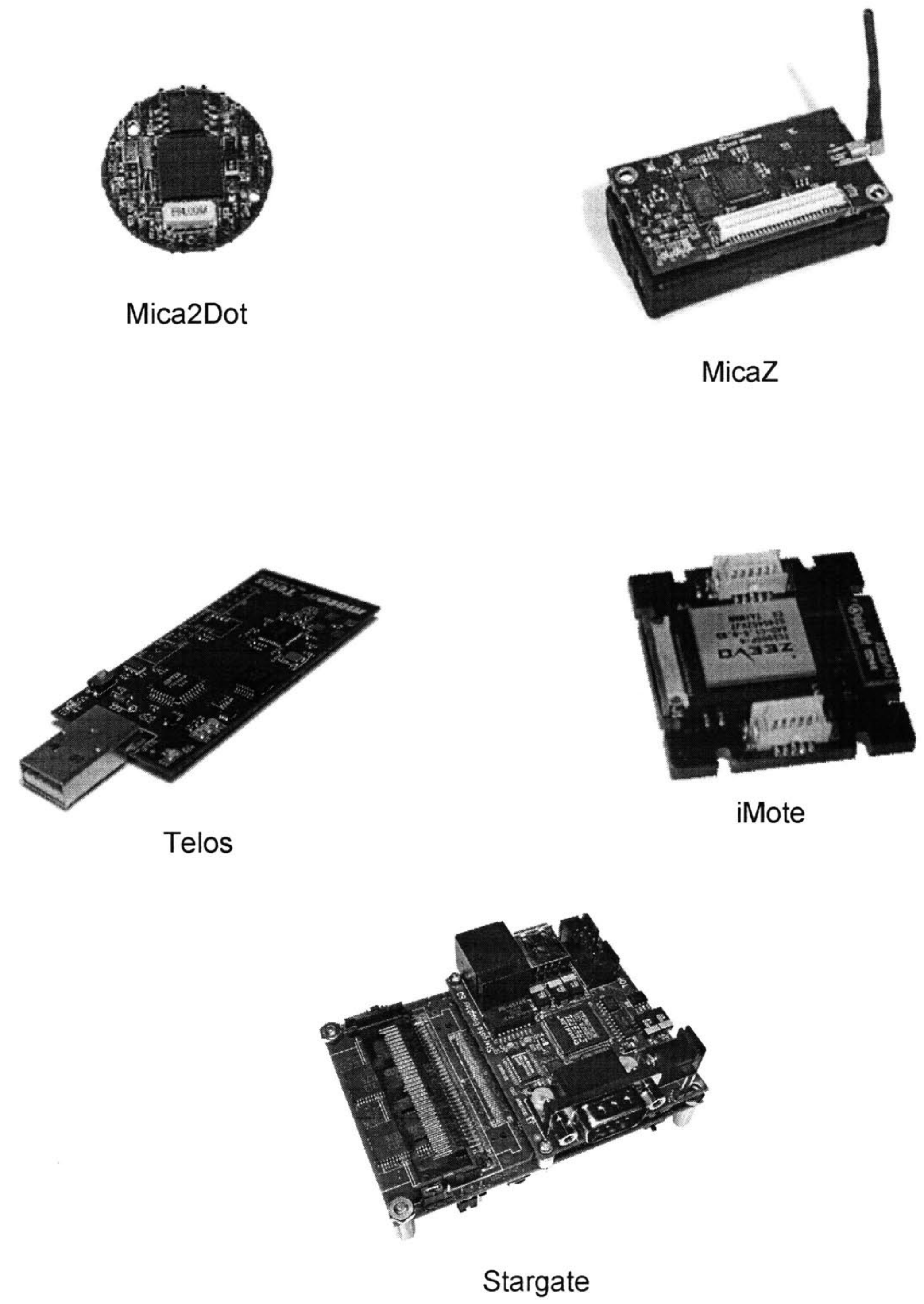

Figure 2-1: Pictured above are the sensor network platforms described in Section 2.1. The Mica2Dot is about the size of a US quarter while the MicaZ and Telos platforms are comparable in size to two AA batteries. The iMote's size falls in between that of the Mica nodes. The Stargate is substantially larger, being about the size of a small paper-back book. Pictures courtesy of [2, 4, 3]

is sized to be mounted against two AA batteries and has a battery life that ranges from weeks to months depending on power management (e.g., using a 1% duty-cycle, motes often last six months). A 512 KByte flash memory chip provides persistent storage, although writing to this chip is about as expensive in terms of energy and time as transmitting over the radio.

The second generation device in this family, the Mica2, upgrades both the processor and radio. The processor is an Atmel ATmega 128L [10], operating at 7.37 MHz with 4 KBytes of RAM and 128 KBytes of flash. The radio is a CC1000 [15] operating at 433 MHz that can transmit at 38.4 Kbits/s. This radio exports a byte-level interface to the main processor, resulting in a much lower radio overhead than that TR1000, which has a bit-level. The Mica2 also comes in smaller form factor, called the Mica2Dot, comparable in size to a US quarter, and can be seen in Figure 2-1.

The latest device in the Mica family is the MicaZ. The most notable change for this version of the product is the upgrade of the radio from the CC1000 to the CC2420 [16]. The CC2420 is 2.4 GHz 802.15.4 [6] ("Zigbee") compliant radio specifically designed for the sensor network community. This new standard provides guidelines for both the physical and MAC layers and reduces host processor overhead by exporting a packet-level interface. Additionally, the CC2420 offers an increase in throughput (250 Kbits/s) without much of an increase in power consumption, allowing developers to create higher rate applications without a substantial reduction in network lifetime. Figure 2-1 shows a MicaZ, which retains the form factor of the Mica2.

2.1.2 MoteIV Telos

While the Crossbow Mica family has been embraced by the research community, many features of its design make it sub-optimal for wide-spread deployment in industrial settings. The MoteIV Telos [42] was designed from the ground-up to address many of the shortcomings of the Mica family.

First, Telos nodes do not include the expensive 51-pin Mote connector. Sensors are integrated directly on the processing board, eliminating the need for a failure-prone expansion board. Second, engineers at MoteIV designed the Telos platform to be low-cost. A node's firmware is downloaded via an inexpensive USB connection rather than an expensive dedicated programming board. Finally the Telos mote was designed with power-management in mind. The Texas Instruments MSP430 microcontroller [48] has a sleep current that is less than the drain rate of an unloaded alkaline battery. Additionally, the wakeup latency from sleep state is sufficiently small (much lower than the Mica), making fine-grain power control more effective. Figure 2-1 shows the physical layout of a Telos node.

2.1.3 Intel Mote

Like Telos, the Intel Mote [26] (iMote) was designed to address many of the deficiencies of the Mica platform. The hardware engineers at Intel interviewed researchers and found that they wanted the following improvements: increased CPU processing power, increased main memory size, improved radio reliability, and a reduction in cost.

The 12 MHz ARM7TDMI processor used in the iMote provides a 4x performance improvement over the original Mica mote. This chip has substantially more memory, with 64 KBytes of RAM and 512 KBytes of flash for program and data storage. For radio communication the iMote uses a Bluetooth [1] radio operating in the 2.4 GHz band at 720 Kbits/s. The Bluetooth protocol also supports many concurrent channels using frequency hopping and can operate worldwide without radio license restrictions. Sensors are attached to the basic processing board via small a on-board connector that interfaces to the I2C bus. Figure 2-1 depicts an iMote prototype.

2.1.4 Intel Stargate

The Intel Stargate represents an different point in the sensor network design space. With a substantially larger footprint, the Stargate combines a relatively robust processor (an Intel PXA-255 400 MHz XScale processor) with ample amounts of memory (64 MBytes RAM, 32 MBytes flash). In addition, multiple network technologies (e.g. 802.11, Ethernet, Mote radios) can be supported via a daughter card and associated PCMCIA slot. This platform is often used as a gateway device for tiered sensor network architectures. From a deployment perspective, the main drawback of the Stargate are its size, energy consumption, and cost. Figure 2-1 shows the Stargate's main processing board.

2.2 TinyOS

The need for a sensor network specific operating system may not be apparent since most microcontrollers can be programmed using C. However, interfacing directly with the hardware can be time-consuming and error prone. More importantly, sensing-oriented applications need only a subset of the system services present in most modern operating systems. In addition, sensor systems are potentially useful to a wide range of disciplines, and consequently, should be very easy to program. Researchers have proposed many light weight operating systems; the operating system that we chose for our implementation, due to its extensive support community, is TinyOS 1.1 [29].

TinyOS is a research operating system developed at U.C. Berkeley for embedded networked sensors. From its inception, TinyOS was designed to be light weight and modular, allowing the code-base to run a wide range of devices. TinyOS uses an event-driven programming model layered on top of a stack-based threading system.

Program execution in TinyOS occurs in the context of tasks and hardware events. Tasks are threads of control that are atomic with respect to other tasks and run to completion. Because there is no notion of kernel or user space in TinyOS, all threads run in the same memory space. The operating system schedules tasks using a simple first-in-first-out (FIFO) queue and are posted by applications to do long running computation. Hardware events occur as a result of interrupts from such components as the radio, sensor, or clock. Hardware interrupts can preempt tasks and are propagated through the OS as events, which may in turn lead to the posting of additional tasks.

The concurrency model in TinyOS prevents blocking and spin loops. Instead, programs rely on *split-phase* execution in which a command is composed of an initial request, which returns immediately, and a later event, which signals the command's completion. This model works well for handling hardware events, but is somewhat inconvenient for high-level operations.

For example, a simple monitoring application might cycle between sleeping, reading from a sensor, and sending out the reading over the radio. In a programming environment that allows blocking, all that is needed is a simple while loop with calls to sleep, sense, and send is all that is needed. Instead, if non-blocking, split-phase function calls are used, the program logic must be transformed into a state machine. Each one of the sleep, sense and send function calls would have an event handler that invokes a next state function. As programs grow in size, split-phase semantics often lead to code bloat and difficult-to-debug state machines.

However, if programs are truly event-oriented, then the split-phase model has many advantages. First, applications can easily be thought of as a set of event handlers. Second, this model allows a single stack to service multiple concurrent activities, reducing RAM requirements. Finally, the threading model in TinyOS requires that tasks cooperatively multi-task. Split-phase programming lends itself well to this type of threading, allowing computation between modules to be interposed along event boundaries.

Programs in TinyOS consist of a graph of components written in NesC [20], which is a dialect of C designed for building sensor network applications. Each component defines a set of interfaces that it `provides` and `uses`. Interfaces consist of a set of `commands`, or down-calls, and a set of `event` handlers, or up-calls. Additionally, a fixed-sized stack frame and a set of *tasks* are associated with each component. Components are *wired* together in a *configuration* file in which interfaces are connected. This wiring of components allows for both the fan-in and fan-out of commands and events (i.e., events are sent to all interfaces wired to handle that event). At compile-time, only those modules and system services defined in the application's configuration file are included in the final binary. All memory requirements for a component are statically determined at compile-time (i.e., the operating system provides no support for dynamic memory allocation).

```
#define TEMPMSG 10
configuration MonitorC {
}
implementation {
  components Main, GenericComm , TimerC, Temp, MonitorM;

  Main.StdControl   -> GenericComm;
  Main.StdControl   -> TimerC;
  Main.StdControl   -> MonitorM;
  Main.StdControl   -> Temp;
  MonitorM.Timer    -> TimerC.Timer[unique("Timer")];
  MonitorM.SendMsg  -> GenericComm.SendMsg[TEMPMSG];
  MonitorM.ADC      -> Temp.TempADC;
}

module MonitorM {
  provides interface StdControl;
  uses {
    interface Timer;
    interface SendMsg;
    interface ADC;
  }
}
implementation {
  TOS_Msg msg;
  bool sendPending = FALSE;
  uint16_t temp;

  command result_t StdControl.init() { return SUCCESS; }

  command result_t StdControl.start() {
    call Timer.start(TIMER_REPEAT, 1000);
    return SUCCESS;
  }

  command result_t StdControl.stop() { return SUCCESS; }

  task void sendMsg() {
    if (!sendPending) {
      msg.data[0] = (uint8_t) temp;
      msg.data[1] = (uint8_t) (temp >> 8)&0xff;
      sendPending = call SendMsg.send(TOS_BCAST_ADDR, sizeof(TOS_Msg),
                                     &msg);
    }
  }

  event result_t Timer.fired() {
    call ADC.getData();
    return SUCCESS;
  }

  async event result_t ADC.dataReady(uint16_t data) {
    temp = data;
    post sendMsg();
    return SUCCESS;
  }

  event result_t SendMsg.sendDone(TOS_MsgPtr pMsg, result_t success) {
    sendPending = FALSE;
    return SUCCESS;
  }
}
```

Figure 2-2: NesC allows developers to create simple sensor applications by wiring together `components` in a `configuration`. The NesC code above is for an application that periodically reports the ambient temperature.

Figure 2-2 shows a sample NesC application for monitoring the temperature in a room. At the top of an application component graph is a *configuration* that defines which components are included in the binary and how these components rely on each other. `MonitorC` is the top-most configuration for this application and declares the components that `MonitorC` needs: `Main` is a system service that initializes all components, `GenericComm` provides the radio stack, `TimerC` provides periodic events, `Temp` provides the interface to read from the temperature sensor, and `MonitorM` is the driver that orchestrates the sleeping, sensing, and sending states of the application. The configuration also creates a mapping between a module that `uses` an interface, and a module that `provides` the interface. For example, `MonitorM` uses the ADC interface to obtain a temperature reading. The ADC interface is implemented by the Temp component; hence, these interfaces are wired together.

`MonitorM` is a simple driver module for our temperature monitoring application. The module begins executing when `Main` calls `StdControl.start`, initializing a periodic timer. Every second, a `Timer.fired` event will be signaled, causing the node to read from the temperature line of its ADC. Notice the split-phase semantics of the `ADC` interface: a call to `ADC.getData` is followed by an `ADC.dataReady` event once the IO is complete. After reading the temperature, `MonitorM` posts a task to send out a message over the radio containing this value. The sleep-send-sense cycle completes when the `SendMsg.sendDone` event is signaled and the node waits for the next `Timer.fired` event.

2.3 Testbed

Simulations, although useful for debugging program logic and modeling simple system dynamics, have many limitations. In particular, simulations of wireless networks can be quite inaccurate due to the complexities of modeling wireless propagation and interference. Consequently, a testbed is crucial for establishing the real-world performance of protocols. To evaluate our congestion control suite, we deployed a 55-node indoor wireless sensor network testbed, as shown in Figure 2-3.

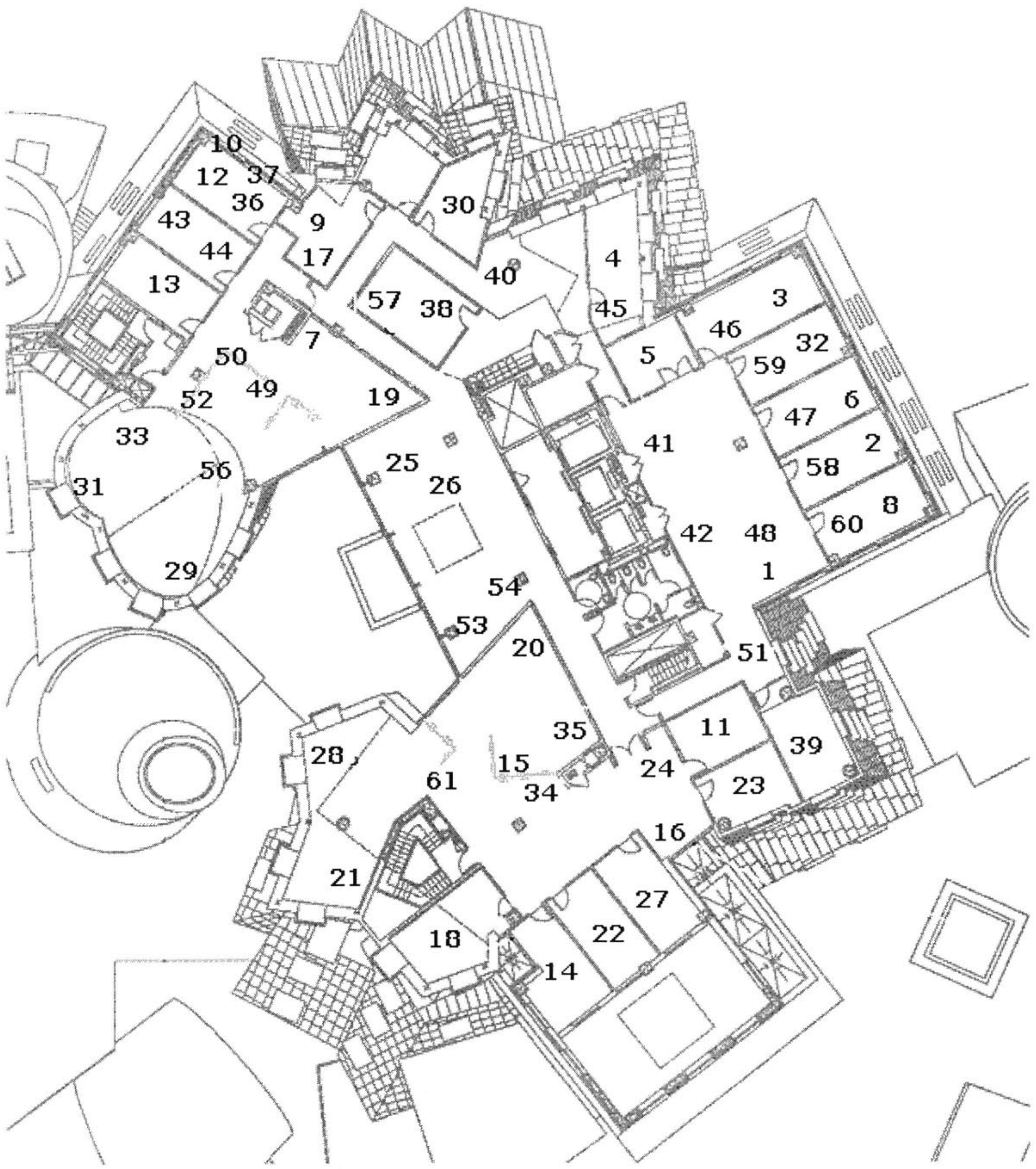

Figure 2-3: This map of our testbed shows the physical locations of the 55 nodes deployed over an area of 16,076 square feet on the ninth floor of the Stata Center at MIT.

2.3.1 Physical layout

The sensor network platform we used in our testbed is the Crossbow Mica2. As described in Section 2.1, a Mica2 node has a 4 MHz, 8 bit microcontroller with 4 KBytes of RAM. Its radio is a CC1000 and can transmit at 38.4 Kbits/s using Manchester encoding[1] [40]. As Figure 2-3 shows, we deployed nodes over an area of 16,076 square feet on the ninth floor of the Gates Tower in the Stata Center at MIT. There is uniform coverage throughout the floor, except for a higher-than-average density in the northwest corner of the floor.

In a typical deployment, a host PC would be used to install each node's firmware one at a time. The nodes would be powered using AA batteries. Statistics would be logged to flash memory on the node for later retrieval. In an active testbed, this evaluation model would require developers to spend a substantial amount of time changing batteries, programming nodes, and retrieving data. Consequently, to reduce the overhead associated with running a sizable deployment, we attach to each node a Crossbow MIB600 interface board. Figure 2-4 shows this setup whereby a Mica2 is attached via an umbilical cable to a MIB600. This so-called "Ethernet programming board" enables developers to remotely monitor, power, and program an attached sensor node. The integrated serial programmer and Ethernet connection allows the testbed to be reprogrammed in under a minute, making sharing the testbed between many projects efficient. Because the MIB600 provides power to the mote (either via a wall outlet or power over Ethernet), there is no need to ever replace batteries. Finally, the Ethernet connection provides a channel through which real-time statistics and debugging messages can be captured from every node. By having an out-of-band communication channel, statistics cannot feasibly be stored on the mote (i.e., packet traces) can be streamed to a central database for processing.

[1]**Manchester encoding** is a convention for representing digital data on a transmission link. A logic 0 is represented using a high to low signal transition and a logic 1 is represented using a low to high transition. Manchester encoding has the nice property that long periods without clock transitions are avoided, making the task of clock synchronization much easier. However, because this scheme doubles the rate at which transitions are made, the encoding is only 50% efficient.

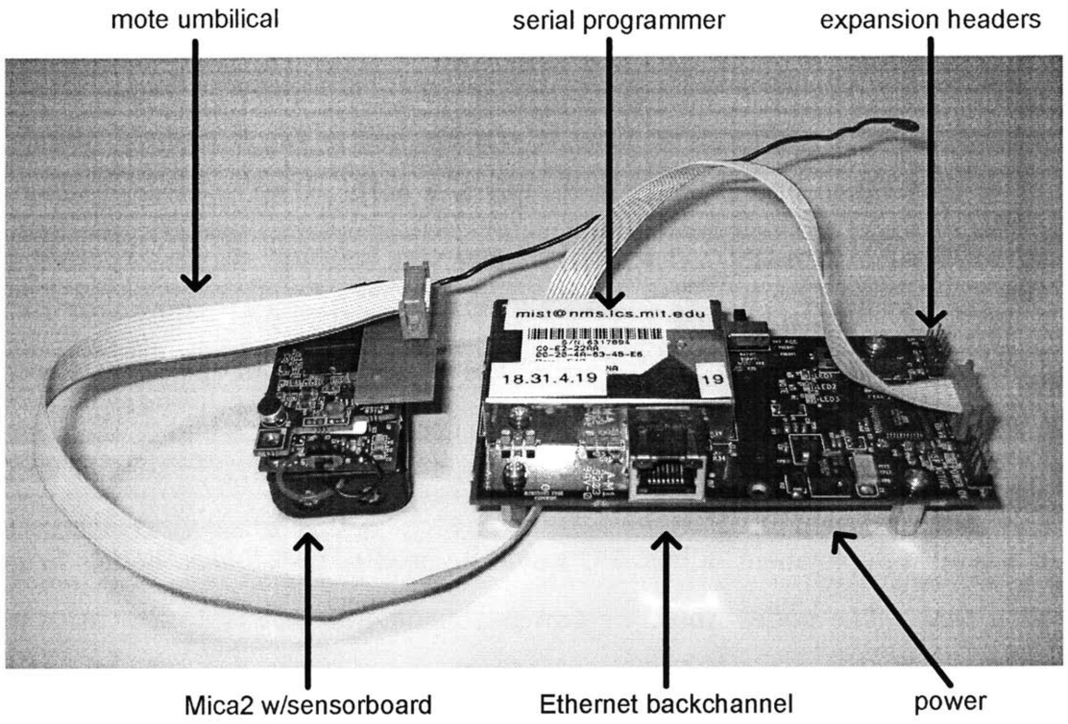

Figure 2-4: Each node in our testbed consists of a Mica2 sensor node connected to a MIB600. The MIB600 provides each node with power, an Ethernet backchannel, and programming capabilities.

Additionally, in order to easily manage and share the testbed between several research projects, we use a management tool called Motelab [5]. This tool provides an integrated suite of Java, Perl, and PHP scripts for user management, web-based scheduling, job management, and centralized data-collection. We made several enhancements to Motelab to increase its reliability and flexibility. Reprogramming is now more resilient to transient network failures and we fixed several bugs in the job scheduling code. In addition, we added a Perl interface to the job scheduler, allowing users to programmatically schedule experiments (i.e., a large number of jobs can now easily be scheduled from a Perl script). Finally, we extended the experiment environment to automatically generate a set of program binaries from the parameters in a configuration file (e.g., to test an algorithm over several radio power levels).

2.3.2 Routing

Wireless sensor network radios are low-power and have limited transmission range in an effort to maximize network lifetime. In order to provide connectivity to geographically-diverse deployments, most wireless sensor networks use a multi-hop routing protocol in which every node forwards traffic. Several general purpose ad-hoc routing protocols exist, including Destination-Sequenced Distance Vector (DSDV) [39], Dynamic Source Routing (DSR) [24], and Ad Hoc On-Demand Distance Vector Routing (AODV) [38]. Although these routing protocols support arbitrary point-to-point communication, sensor network communication patterns tend to fall into two categories: *neighborhood-oriented* (e.g., link-state protocols) or *many-to-one* (e.g., monitoring and archival at a central node).

In our testbed we use a many-to-one topology where all nodes forward readings to a data collection point called a *sensor access point*, or SAP. Each node uses a packet queue of size eightto buffer traffic as it forwards data along a spanning tree rooted at the SAP. This spanning tree is built using our own implementation of the the DSDV routing protocol, selected for its simplicity and modest memory footprint.

DSDV is a hop-by-hop distance vector protocol in which nodes maintain a routing table containing entries for every destination in the network. Each entry in the routing table contains the destination's identifier, the next hop to reach to the destination, the destination's last heard sequence number, and a metric. The routing protocol uses sequence numbers to ensure that routes are fresh and metrics to select between multiple routes for a given destination. Periodically, nodes will broadcast an advertisement containing their complete routing table. In our implementation, we only support a single destination (the SAP) because the workloads we analyze in Chapter 5 do not require arbitrary point-to-point connectivity.

The metric that nodes in our testbed use to choose paths to the SAP is called ETX [17]. ETX measures the expected number of transmissions required to deliver data to a destination. The ETX of a link is calculated using the forward and reverse delivery rates of a link, d_f and d_r:

$$\mathrm{ETX}_{\mathrm{link}} = \frac{1}{d_f \cdot d_r} \tag{2.1}$$

In order to calculate delivery rates for a link, each node monitors the channel and maintains a neighbor table that records delivery rates. Every packet header transmitted from a given source contains an increasing sequence number. By overhearing a neighbor's outgoing traffic, a node calculates the delivery rate from that node to itself based on the gaps in the overheard sequence numbers. Periodically, each node broadcasts its complete link quality table, allowing nodes in the neighborhood to collect bi-directional delivery rates for each next hop.

A path's ETX is simply the sum of the ETX values over the links from source to destination. In our testbed, a node advertises an ETX for the complete path to the SAP in every DSDV routing beacon. Other nodes in the neighborhood calculate the end-to-end ETX by adding this advertised ETX to their own a one-hop ETX calculation for the link to the node that sent out the beacon. The node then selects a path that minimizes the end-to-end ETX.

2.3.3 Medium access control

Network performance in wireless systems is significantly affected by the *medium access control* (MAC) protocol. In our testbed we use an enhanced version of B-MAC [41], a carrier sense medium access protocol for wireless sensor networks distributed with TinyOS (our modifications are described in Section 4.4). B-MAC uses clear channel assessment (CCA) to determine when the channel is idle and it can transmit a packet. To improve the accuracy of CCA, B-MAC employs automatic gain control for estimating the noise floor. Automatic gain control operates by periodically sampling the energy level of the channel when it is idle and feeding those readings into an exponentially weighted moving average estimator.

In B-MAC, all packet transmissions are preceded by an 18 byte preamble and a 2 byte synchronization sequence. In TinyOS, the default size of a packet is 36 bytes, bringing the total number of bytes transmitted to 56 bytes. Therefore, the Mica2, which operates at 38.4 KBytes/s and uses Manchester encoding, take about 24 ms to transmit a packet once it gains access to the channel.

B-MAC provides a power saving mechanism called Low Power Listening (LPL). By periodically sampling the channel and only fully waking-up the node when the radio detects activity, LPL dramatically increases node lifetime. In order for LPL to work properly, packet preambles must be as long as the check interval, which ranges from 10 ms to 1600 ms. In our testbed, for simplicity, we do not use LPL. We leave a detailed evaluation of the effects of power management on congestion control for future work.

Per-hop data delivery reliability in B-MAC is improved using link-level acknowledgements. At the end of every successful unicast packet reception an acknowledgment code is sent from the receiver back to the sender. Applications are informed as to whether or not this acknowledge code was received, allowing them to retransmit if necessary. In our testbed, packets are retransmitted up to three times (i.e, a maximum of four transmissions per packet).

2.3.4 Radio range

On factor that greatly affects the average neighborhood size (and routing topology) of our testbed is radio power. Characterizing the size of each node's neighborhood is difficult because radio signals are time-varying. To characterize neighborhood size, we measure channel quality between all pairs of nodes in an unloaded network for several different transmit power levels. Note that we perform this measurement without a routing protocol or any other network stack modifications. One-by-one, each node sends a train of broadcast probe packets (without link-level retransmissions). We define the size of a node's neighborhood $N(x)$ to be the expected number of nodes that will hear any given transmission. This value can be calculated using Equation 2.2, where x is any node in the network, $\mathcal{N}$ is the set of all nodes, and p_{xy} is the probability node y successfully receives node x's transmission (without any retransmissions).

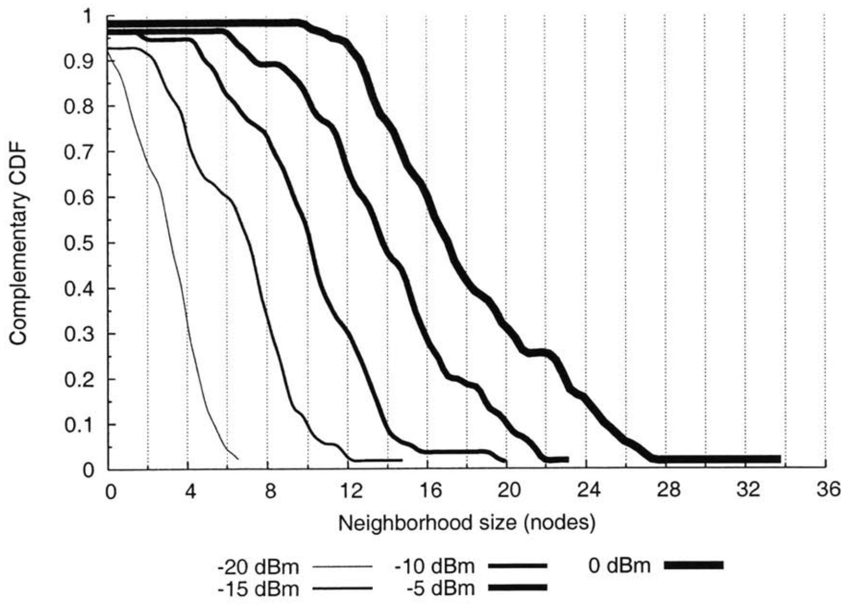

Figure 2-5: Neighborhood size (complementary CDF) as computed from Equation 2.2 for different transmit power levels in the 55-node indoor wireless sensor network testbed depicted in Figure 2-3.

$$N(x) = \sum_{y \in \mathcal{N} - \{x\}} p_{xy} \tag{2.2}$$

Figure 2-5 shows node neighborhood sizes in our testbed. Note that the average neighborhood size increases linearly with an exponential increase in power. For our experiments, we selected a transmit power level of -10 dBm, which is significantly lower (by a factor of 10) than the default power level of 0 dBm. By reducing the default power level, we hope to reduce radio contention and increase spatial reuse while maintaining a connected network.

2.4 Related work

As mentioned in Chapter 1, much of the work presented in this thesis builds on ideas from many other researchers. What follows is a brief discussion of some of the work that most closely relates to congestion control in wireless sensor networks.

Wan *et al.* propose Congestion Detection and Avoidance (CODA) [52], a congestion control system for sensor networks. CODA detects congestion by periodically sampling the channel load and comparing the fraction of time that the channel is busy to the maximum channel utilization. Maximum channel utilization is calculated offline using a well-known formula[2] [13] for the capacity of CSMA networks coupled with platform specific measurements of the channel idle detection delay for the Rene2 mote. The system responds to congestion with a combination of hop-by-hop flow control and closed-loop regulation. In our work, we experimentally evaluate CODA's congestion detection mechanism (channel sampling) and one of its congestion mitigation mechanisms (hop-by-hop flow control). We make all our comparisons in a large sensor network testbed, expanding on previous small-scale testbed or simulation-based congestion control studies. We find that when used alone, channel sampling-based congestion detection performs worse than queue occupancy-based congestion detection. We also find that augmenting a hop-by-hop flow control mechanism (such as CODA) with rate limiting is beneficial.

Woo and Culler propose a rate control mechanism [54] that admits traffic into the network using an additive-increase multiplicative-decrease (AIMD) controller. When a node overhears that a packet it had previously sent was forwarded, it additively increases its transmission rate. When it does not hear a previous transmission being successfully forwarded (presumably after a timeout), it multiplicatively reduces its transmission rate. We evaluated a similar rate control mechanism. We found that rate limiting increases fairness, but its benefits to the network (as measured by the "efficiency" and "fairness" metrics from Chapter 3) are most significant when used in combination with other congestion control techniques.

Lu *et al.* propose RAP [30], a real-time communication protocol for sensor networks. The network layer of RAP ensures fairness between nodes and improves the

[2]CODA uses the following formula to approximate the maximum throughput of a channel using CSMA:

$$S_{\max} = \frac{1}{1 + 2\sqrt{\frac{\tau C}{L}}} \tag{2.3}$$

where τ is the channel idle detection delay in seconds, C is the raw channel bit rate, and L is the expected number of bits in a data packet.

ratio of packets that make their latency deadlines. To accomplish this task, RAP provides support for deadline- and distance-aware packet scheduling. Packets originating from sources deeper in the network have higher priority than packets originating from sources close to the sink. While RAP focuses on the network's ability to meet deadlines, our work focuses on managing overload and congestion in a sensor network.

Sankarasubramaniam *et al.* propose ESRT [44], the Event-to-Sink Reliable Transport Protocol. Their system addresses congestion control in the context of reliable delivery. ESRT attempts to keep a network operating near its optimal load by broadcasting one-hop control messages to sources from the sink. The consequent assumption is that a data sink can reach all sources via a high-powered one-hop broadcast, which reduces overall network capacity. In contrast, our hop-by-hop flow control does not require a high-powered broadcast message to be sent by a sink.

Ee and Bajcsy propose alleviating congestion and achieving fairness in many-to-one wireless sensor networks through a combination of rate limiting and per-child queuing in tree-based network topologies [18]. Their rate limiting approach involves estimating the average rate at which packets can be sent from a node and dividing that between the total sub tree that a given node supports. Rates are piggy-backed on the headers of all outgoing packets and nodes overhear each other's traffic, ensuring that children do not send at a rate exceeding their parent's capacity. Additionally, the authors propose Epoch-Based Proportional Selection to service per-child queues to ensure fairness. This rate limiting algorithm could easily be used as part of the congestion control suite we propose in Chapter 4. However, our proposed method of source rate limiting requires does not require additional packet headers nor does it rely per-child queuing.

Yi and Shakkottai propose a fair hop-by-hop congestion control algorithm for multi-hop wireless networks [59] using cost functions. They build a theoretical model and provide a simulation-based evaluation of their distributed algorithm. They make the assumption that simultaneous transmissions can occur over links in the same radio neighborhood, using orthogonal code division-multiplexing channels. Such approaches require sophisticated code management algorithms. In the sensor networks we analyze, all nodes operate at the same frequency, and hence concurrent successful transmissions within the same radio neighborhood usually do not occur.

Lemmon *et al.* study overload in sensor-actuator networks connected by a wired bus [28]. The key difference between their overload problem and ours is that the communication network they consider is a shared bus, with no potential for spatial reuse. Additionally, their sensor nodes do not forward each other's traffic.

Zhao and Govindan conduct a comprehensive study of packet delivery performance in wireless sensor networks [60]. Their study focuses on the physical and link layers, evaluating packet loss, packet loss correlations, and link asymmetry. Our study of congestion complements their work, studying end-to-end performance when sensors participate in multi-hop routing and congestion avoidance protocols. Our congestion control algorithms operate in a network with a wide range of link loss rates and asymmetries, motivated in part by their results.

Woo *et al.* examine routing in sensor networks [55], studying link estimation and neighborhood table management in particular. We use these mechanisms in our network layer implementation to support our congestion control algorithms.

Hop-by-hop flow control protocols have been extensively studied in the context of ATM and local-area networks [27, 34, 36, 37]. The motivation in these high-speed networks is to avoid the burst behavior of end-to-end protocols like TCP at small round-trip times. In sensor networks, hop-by-hop flow control is attractive because it allows good congestion adaptation without incurring losses or requiring the expensive end-to-end acknowledgments that are unnecessary for many streams that don't require TCP-style reliability. Section 3.3 discusses in greater detail why TCP is not appropriate for wireless sensor networks.

Chapter 3

The Congestion Problem

Congestion in wireless sensor networks, as in the Internet, can have disastrous consequences. However, the symptoms and causes of congestion in wireless sensor networks are quite different from the Internet. In wired networks loss typically manifests itself as buffer drops at internal routers, and protocols such as TCP alleviate congestion. In wireless networks loss occurs both at transit nodes as well as in the transmission medium. Moreover, current methods of congestion control are inappropriate for this domain. In Section 3.1 we investigate the symptoms and quantify the effects of congestion on a wireless sensor network. In Section 3.2 we discuss several reasons for the degradation in performance. Section 3.3 addresses why TCP, which alleviates congestion on the Internet, is not an appropriate congestion control mechanism for sensor networks. Finally, in Section 3.4 we present several metrics to evaluate network performance that we will use in the remaining chapters.

3.1 Congestion symptoms

This section diagnoses two key symptoms of congestion collapse in wireless sensor networks. The following results are derived from the indoor Mica2 wireless sensor network testbed, described in Section 2.3. In this network every node generates data at a constant rate, which nodes forward over a multi-hop network to a single sink.

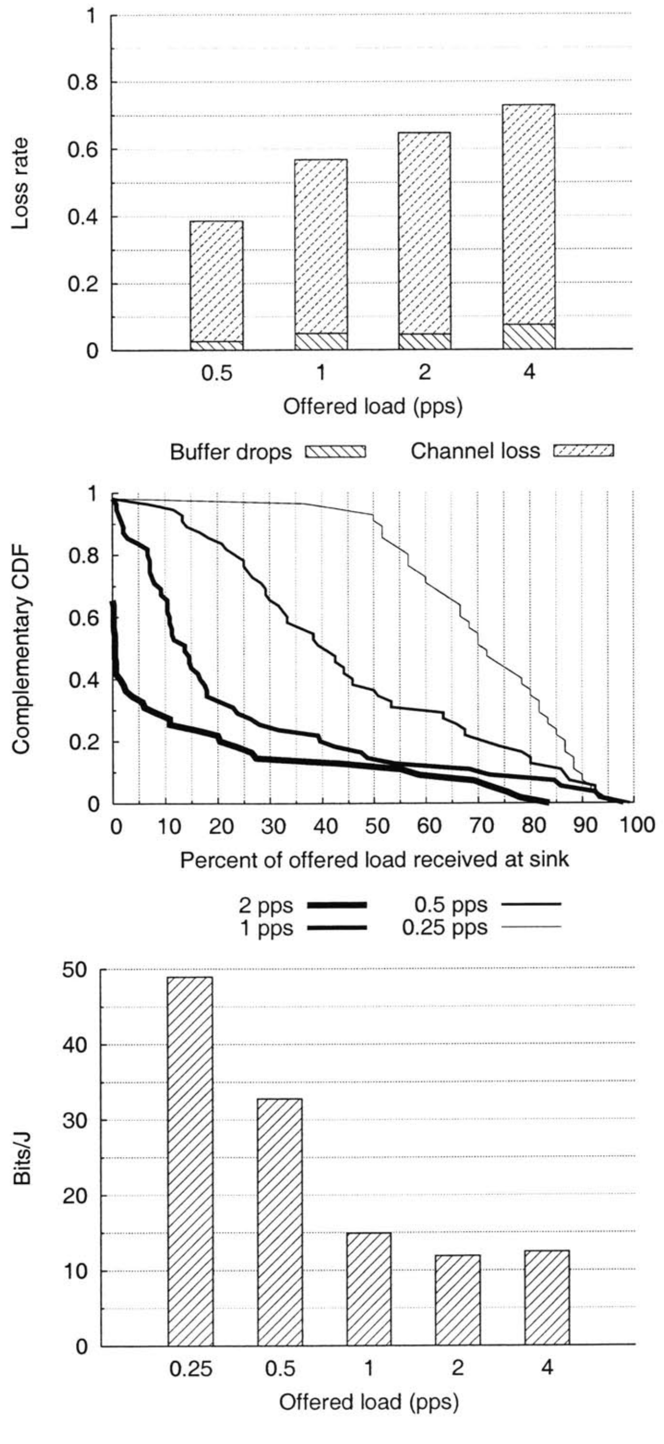

Figure 3-1: The above three plots show the effects of congestion on loss rate, fairness, and energy consumption for our testbed deployment. The *top* plot shows channel and buffer loss rate as a function of per-node offered load. Note how wireless drops dominate and increase substantially as load increases. The *middle* plot shows the percentage of each node's offered load that is received at the sink. As offered load increases, the network becomes increasingly unfair, with a few nodes delivering the majority of the traffic. The *bottom* plot shows network-wide number of bits successfully transmitted per unit energy. The energy efficiency of a network declines with the onset of congestion.

As the offered load increases, loss rates quickly increase. Figure 3-1 (*top*) shows the network-wide packet loss rates for various offered loads. We separate losses due to wireless channel errors from losses caused by a lack of buffer space at a sensor node. We see that channel losses dominate buffer drops and increase quickly with offered load. This dramatic increase in loss rates is one of the two symptoms of congestion collapse. Congestion control schemes must decrease the channel loss rate in order to reduce the number of wasted transmissions.

The second symptom of congestion collapse is the starvation of most of the network due to traffic from nodes one hop away from the sink. Figure 3-1 (*middle*) illustrates this phenomenon. Given a percentage of packets p received from a given node at the sink, the complementary CDF plots the fraction of sensors that deliver at least p percent of their data to the sink. We see that as the offered load increases, a decreasing number of nodes get a disproportionately large portion of bandwidth. For example, at 2 packets/s, close to 70 percent of the nodes are experience severe starvation and achieve only 10 percent of their offered load. If we were providing fire protection in a ten-story building, this network would only provide adequate coverage for the first three floors. Reducing the number of nodes that achieve less than 10% (or some other threshold depending on the application) of their offered load is crucial for creating a usable network.

Congestion collapse has dire consequences for energy efficiency in sensor networks, as Figure 3-1 (*bottom*) shows. When offered load increases past the point of congestion, fewer bits can be sent with the same amount of energy. The network wastes energy transmitting bits from the edge towards the sink, only to be dropped. We call this phenomenon *livelock*, defined as the situation in which a network transmits increasing amounts of traffic that never reach an access point.

3.2 Sources of channel degradation

There are two principal ways in which channel quality degrades in wireless networks: through the overlapping transmissions of *hidden terminals* and through an increase

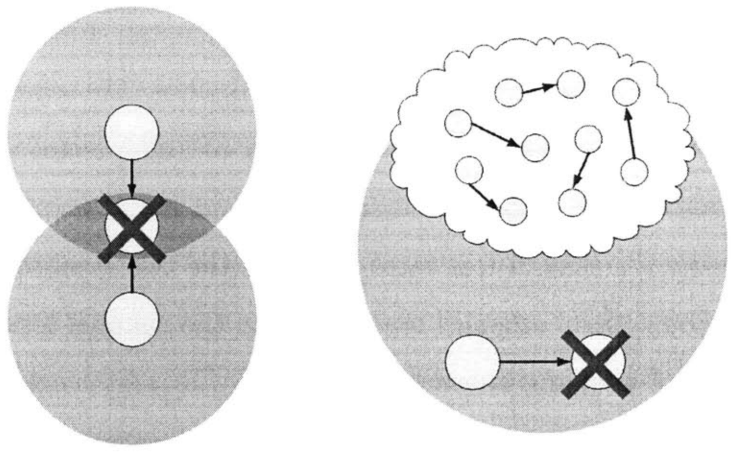

Figure 3-2: In our network there are two sources of transmission loss: hidden terminals and interference. The *left-hand* side of the figure depicts an instance of the hidden terminal problem involving two senders that cannot hear each other transmit to a common receiver resulting in a collision. The *right-hand* side of the figure shows packet loss due to the additive effects of many distant senders raising the noise floor and corrupting packet transmissions.

in the *noise floor* caused by many distant, simultaneous transmissions. We explore these two problems in the next sections.

3.2.1 Hidden terminals

Hidden terminals occur when two or more transmitting nodes that are outside of each other's radio reception range share a common receiver. The left-hand side of Figure 3-2 shows this scenario where the top and bottom nodes are the hidden terminals. If both of the hidden terminals in this figure were to transmit a packet, the middle node would not receive either packet due to interference caused by the overlapping transmissions.

The prevalence of hidden terminals in wireless networks depends in part on topology. In wireless sensor networks, nodes often communicate with an access point via a multi-hop spanning tree. Nodes usually select the path to the access point using a metric derived from link quality. Consequently, levels of the forwarding tree tend to be based on radio neighborhoods. Often, a node is a hidden terminal with respect to its grandparent, and sometimes its siblings. It is important to remember that in a deployed network, the location of a hidden terminal varies with time because a hidden

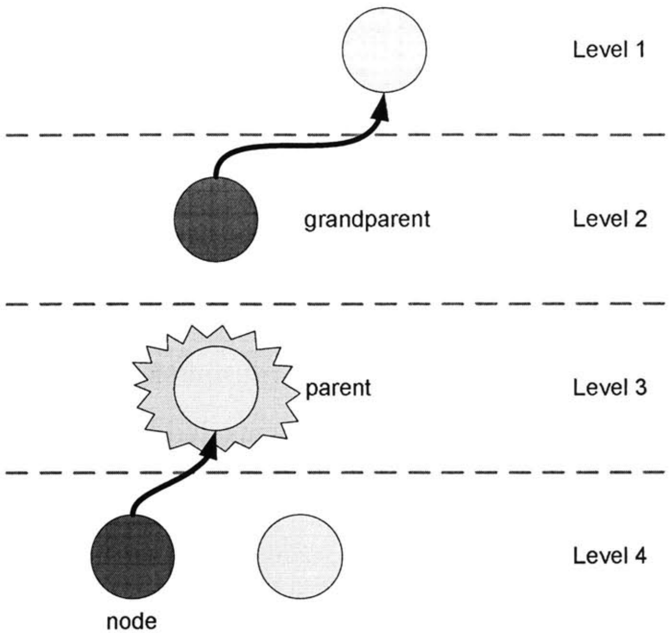

Figure 3-3: In tree-based routing topologies hidden terminals naturally occur between alternating levels of the forwarding tree. In this figure, the bottom node (Level 4) is a hidden terminal with respect to its grandparent (Level 2). If this node transmits immediately after its parent forwards a packet—meaning its grandparent is likely be forwarding as well—a collision will likely occur at the parent.

terminal's existence depends on reception range (which varies with time). Two nodes connected by a marginal link may be hidden terminals with respect to each another only part of the time.

Figure 3-3 shows a configuration in which a node risks a *collision* if it transmits to its parent when its *grandparent* is forwarding packets (which the node cannot hear). Alternating levels of multi-hop forwarding trees are particularly prone to this problem.

Additionally, the susceptibility of a network to hidden terminal collisions is closely tied to its MAC protocol. Carrier sense multiple access (CSMA) based MAC protocols are popular in sensor networks due to their simplicity and work-conserving[1] properties. Unfortunately, CSMA MACs are particularly prone to hidden terminals because the choice of when to send is based on channel conditions at the transmitter rather than

[1]We use the term **work-conserving** to describe protocols that allow data to be sent any time the medium is idle. CSMA is a work-conserving MAC protocol because any node is allowed to transmit when the channel is idle. TDMA [53] is an example of a non-work-conserving MAC protocol because a node can only transmit in its assigned slot and idle slots are not redistributed to nodes with data to send.

at the receiver, which is where the collision would occur. Higher level mechanisms can be layered on top of a CSMA MAC to reduce hidden terminal collisions. For example, every transmission could be preceded by a "virtual carrier sense handshake" [25]. In this scheme, initially a *request to send* (RTS) packet is broadcast by the transmitter; the receiver responds by sending a *clear to send* (CTS) packet. Once the transmitter receives a CTS, it begins transmitting the data packet. Other nodes hearing the CTS do not send until the previously started transmission is completed. Exchanging RTS/CTS control packets is effective at reducing hidden terminals. However, the overhead associated with the additional transmissions is only worthwhile if the control packets are much smaller than the data packets. In sensor networks, where data packets tend to be small, the exchange of RTS/CTS packets is generally considered to be too heavy weight of a mechanism to be efficient.

3.2.2 The noise floor

Packet loss also occurs due to interference caused by the aggregate affect of many distant transmissions. The right-hand side of Figure 3-2 shows this effect graphically.

The noise floor refers to the energy level present on the channel that cannot be decoded into a packet with a valid CRC. This ambient energy level can be thought of as the background noise in room. In an empty room, the noise floor is simply the white noise of the environment. As the room becomes crowded, the aggregate affect of many people talking makes its increasingly difficult for any two people to hold a conversation. Likewise, in the wireless domain, the additive effect of many distant senders can cause the noise floor to rise, making it difficult for a receiver to separate signal from noise.

Note that this problem differs a hidden terminal collision. With hidden terminals, transmission zones overlap at a common receiver. In contrast, corruption due to an increase in the noise floor results from the additive effects of nodes outside the transmission range.

Reducing the effects of a rise in the noise floor is difficult because interference is not caused by any single node. One strategy that attempts to address these issues is rate limiting, which we explore Chapters 4 and 5.

3.3 Why not TCP?

On the Internet, congestion control is done end-to-end in transport protocols like TCP or in end-system modules like the Congestion Manager [12]. End-to-end flow control schemes like TCP are not well-suited to our domain for the following reasons:

1. *Congestion detection:* TCP uses packet loss to detect congestion. This heuristic works well in wired networks where transmission losses are rare and buffer drops are the primary cause of loss. When a packet is dropped, TCP assumes its transmission rate is too high, causing buffers along the path to the destination to overflow. However, this inference is incorrect in wireless networks where loss predominantly occurs in the transmission medium.

2. *Data generation mismatch:* Many sensor network applications involve low, constant bit rate (CBR) flows that might experience a sudden increase in transmission rate when an interesting event occurs. With TCP, every incoming ACK causes an increase in the transmission window size. The problem is that if the stream was not saturating the network (as in a low-rate CBR), this window inflation is artificial and does not signify that the capacity indicated by the window is actually available. When an event occurs that causes a sequence of packets to be sent in quick succession, TCP would assume that the large window was usable, but the result would be packet loss because the network isn't actually capable of sustaining this large window (rate).

3. *End-to-end acknowledgment overhead:* Many end-to-end congestion control schemes require ACKs to be sent from the receiver, to allow the sender to obtain an accurate idea of the state of the network. Many sensor streams don't

require the reliability semantics of TCP, making the cumulative ACKs unnecessary. Furthermore, since most sensor data packets are small, end-to-end ACKs would consume a substantial fraction of the overall network bandwidth.

4. *Small window performance:* Protocols like TCP are notorious for poor performance when windows are small [11], as would be the case in many low-bandwidth sensor networks. Although some recent solutions have been proposed for this problem [8], a fundamental problem is that small-window paths often tend to cause high packet loss rates in the way TCP adapts while probing for more bandwidth.

While TCP could possibly be modified to accommodate sensor networks, these four points show that wireless sensor network design constraints and traffic patterns necessitate a different model for congestion control. In Chapter 4 we discuss several techniques for controlling congestion that are more appropriate for the sensor network domain.

3.4 Metrics

Based on these quantitative observations, we propose a number of metrics to evaluate the performance of sensor networks under congestion: *network efficiency* η, *node imbalance* ζ, *network fairness* ϕ, *aggregate sink received throughput*, and *median packet latency*. Table 3.1 summarizes these metrics.

We define *efficiency* η as the number of hops "useful" packets travel, divided by the total number of packet transmissions in the network (Equation 3.1 in Table 3.1). A *useful* packet is any packet that eventually reaches its destination and its usefulness is weighted by the distance it travels in hops. Efficiency is important to measure because network bandwidth is limited, and energy is often a scarce resource in sensor networks. Motivated by the previous energy result (Figure 3-1, *right*), this definition extends previous notions of link efficiency [60, §5.3] to a multi-hop sensor network setting. The denominator of the metric (total number of transmissions) includes all

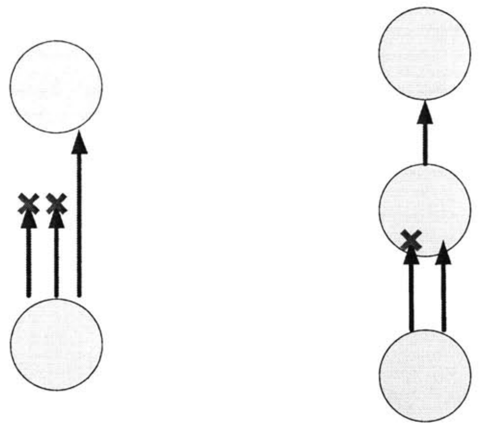

Figure 3-4: Efficiency is the ratio of the number of packets received at the sink weighted by the distance each packet travels to the total number of transmissions. In the *left-hand* diagram, node A sends one packet to B, but because of channel loss, B must retransmit twice. The efficiency η for this network is $\frac{1}{3}$: 1 packet received over 1 hop, with a total of 3 transmissions. In the *right-hand* diagram, node A sends 2 packets to B over 2 hops, 1 of which is dropped. The efficiency for this network is $\frac{2}{3}$: 1 packet received over 2 hops, with a total of 3 transmissions.

retransmissions, as well as transmissions associated with packets that are eventually dropped or corrupted.

One of the primary motivations for our efficiency metric is that a packet originating far from the sink should *not* be penalized for being forwarded through a greater number of hops. This goal is achieved by weighting the utility of a successfully delivered packet by the number of hops it travels. In addition, our efficiency metric penalizes failed transmissions, buffer drops, retransmissions due to lost acknowledgments, and channel losses. Efficiency penalizes a dropped packet to a greater extent if it travels a greater number of hops toward a sink because these packets are costly for the network in terms of contention and energy. Efficiency therefore measures the *fraction of transmissions in a sensor network that contribute to a packet's eventual delivery at the sink.* Efficiency also measures the fraction of transmissions whose energy is not wasted. Figure 3-4 shows two examples of how efficiency is computed.

Related to efficiency is *imbalance*, a per-node measure of how well each node forwards data. We define the *imbalance* ζ *at node* i using Equation 3.2 in Table 3.1. Note, that for the tree-based forwarding topologies, $N(i)$ contains all nodes that were i's parent at some point in time. Figure 3-5 shows an example of how we calculate

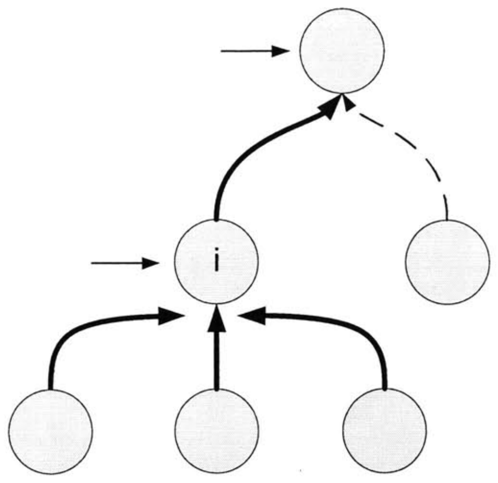

Figure 3-5: Imbalance measures the amount of traffic that a node successfully forwards. In the above diagram, node i receives 6 packets from its children. It forwards these packets to its parent, but only 3 are received (3 packets are lost either in the channel or dropped by the parent's queue). Therefore, the imbalance for node i is $\zeta = \frac{6}{3} = 2$.

imbalance. When $\zeta = 1$ for node i, i receives and transmits equal amounts of data from its neighbors to its next hop nodes. A large ζ indicates that node i receives more data from its neighbors than it successfully transmits. Imbalance differs from a simple buffer drop count, because it is directly influenced by wireless drops. Imbalance is particularly useful for identifying "hot-spots" in the network that account for a sizable fraction of the losses in efficiency.

Achieving *fairness* is desirable because in many sensing applications there is a decreasing marginal utility of increasing a sensor's report rate. In other words, it is often more important to hear a low rate of traffic from N sensors spread out across a large area than a high rate from one sensor. Achieving fairness in multi-hop wireless networks is difficult and often comes at the cost of reduced aggregate throughput [31, 35]. We measure fairness ϕ with the index [23] shown in Table 3.1.

While efficiency and imbalance capture how efficiently a sensor network delivers data in terms of its use of transmission opportunities and energy, they do not measure the overall rate of data delivery. We therefore report aggregate and median per-node throughput, both measured at the sink.

Metric	Definition	Parameters
Efficiency	$$\eta = \frac{\sum_{u \in U} \mathbf{hops}(u)}{\sum_{p \in P} \sum_{h \in \mathbf{hops}(p)} \mathbf{xmits}(p, h)} \quad (3.1)$$	U is the set of useful packets, P is the set of all packets, $\mathbf{hops}(p)$ ranges over each hop packet p takes, and $\mathbf{xmits}(p, h)$ counts the number of transmissions packet p undergoes at hop h.
Imbalance	$$\zeta = \frac{\#\text{ pckts rcvd at } i}{\sum_{j \in N(i)} \#\text{ pckts rcvd at } j \text{ from } i} \quad (3.2)$$	This metric is defined for each i; packet counts are taken over the entire experiment. In our expts., $N(i)$ is the set of parents for node i.
Fairness	$$\phi = \frac{\left(\sum_{i=1}^{N} r_i\right)^2}{N \sum_{i=1}^{N} r_i^2}. \quad (3.3)$$	The average rate of packets delivered from the ith sensor is denoted r_i. N is the number of sensors in the network.
Throughput		Measured aggregate and median per-node throughput received at the sink.
Latency		Median latency of packets received at the sink.

Table 3.1: The metrics we use to evaluate network performance.

Median packet latency is also important to measure because many applications require that the time between sensing and reporting be minimal. Traffic overload in a sensor network often increases latency. It is important to note that although adding buffering and flow control helps to alleviate congestion, it also increases the queuing delay of packets at each hop, increasing end-to-end latency.

We use these five metrics in Chapter 5 to evaluate the performance of our congestion control techniques in our testbed.

Chapter 4

Mitigating Congestion

In Chapter 3 we saw how congestion degrades channel quality, increases energy consumption, and leads to node starvation. In this chapter we propose several techniques for mitigating congestion in sensor networks. These techniques have two explicit goals:

1. *Increase network efficiency* to reduce energy consumption and improve channel quality.
2. *Avoid starvation* to improve the per-node end-to-end throughput distribution.

In the first section we present a high-level overview of our scheme, *Fusion*, and then follow with several sections describing its component mechanisms.

4.1 Fusion: An integrated approach to congestion

The congestion control scheme we propose is called *Fusion* and integrates three techniques: hop-by-hop flow control, rate limiting, and a prioritized MAC. Hop-by-hop flow control is designed to prevent nodes from transmitting if their packets are only destined to be dropped due to insufficient space in output queues at downstream nodes. Rate limiting meters traffic being admitted into the network to prevent unfairness toward sources far from a sink. A prioritized MAC ensures that congested nodes receive prioritized access to the channel, allowing output queues to drain. While these techniques do not explicitly rely on topology information, we focus on single-sink, spanning-tree topologies in this thesis.

We also note that the application of these techniques is difficult in the wireless domain. This difficulty arises from the fact that contention for the wireless channel occurs at both the sender and the receiver. Indoors, this contention is particularly problematic, since radio reflection makes propagation erratic and causes interference between two seemingly-disjoint sets of nodes. Additionally, there is a natural trade-off between channel utilization and fairness. By allowing nodes that provide transit for large amounts of traffic to transmit, we essentially allocate bandwidth to those nodes with the greatest contention. Finally, the wireless channel is inherently lossy, making distributed control of data flows even more challenging.

4.2 Hop-by-hop flow control

The first component of Fusion we discuss is hop-by-hop flow control, which has been proposed in wired local-area and wide-area networks [34, 36, 37], as well as in sensor networks [30, 52]. In our implementation, each node sets a *congestion* bit in the header of every outgoing packet. By taking advantage of the broadcast nature of the wireless medium, our implementation provides congestion feedback with every transmission to all nodes in a radio neighborhood. As a result, this implicit feedback obviates the need for explicit control messages that could use a significant fraction of available bandwidth.

Hop-by-hop flow control has two components: congestion detection and congestion mitigation. We first discuss several methods of detecting congestion: using queue occupancy, channel sampling, or packet loss.

A simple way to detect congestion relies on monitoring a node's queue size: if the fraction of space available in the output queue falls below a high water mark α (in our implementation, $\alpha = 0.25$), the congestion bit of outgoing packets is set; otherwise the congestion bit is cleared. This technique, which we evaluate in Chapter 5 as *queue occupancy*, incurs little additional overhead.

CODA [52] proposes an alternate way to detect congestion. When a packet is waiting to be sent, the node samples the state of the channel at a fixed interval.

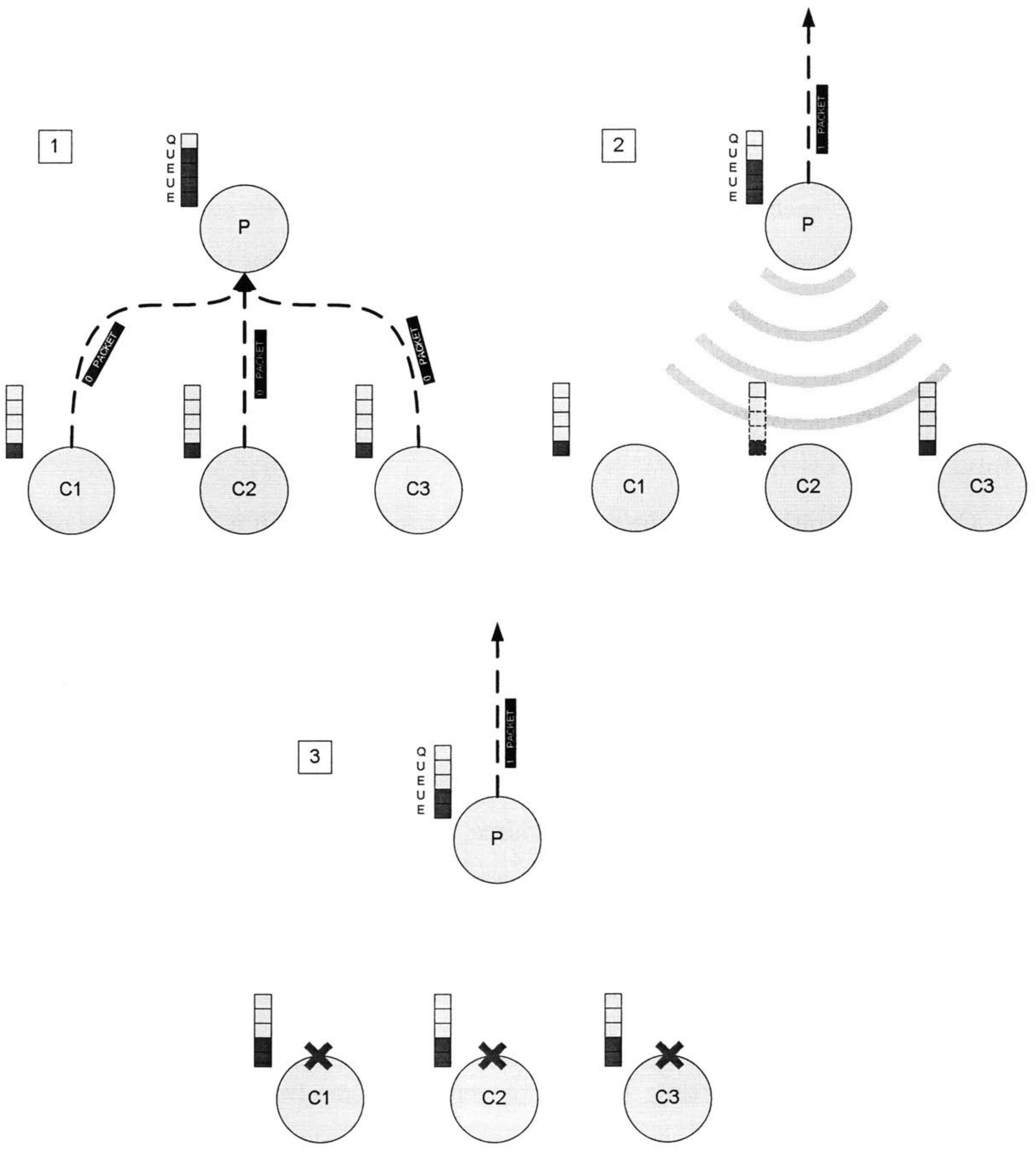

Figure 4-1: Hop-by-hop flow control provides feedback on a link by link basis. *Step 1* shows a scenario where three nodes (C1, C2, and C3) simultaneously forward traffic to a common parent (P). Once queues build up at node P, it sets the congestion bit in all outgoing traffic, which other nodes monitor (*Step 2*). When nodes C1, C2, and C3 hear that their next hop node P has set its congestion bit, they stop sending, allowing node P to drain its queues (*Step 3*).

Based on the number of times the channel is busy, it calculates a utilization factor. If utilization rises above a certain level, it sets the congestion bit. Otherwise, the node clears the congestion bit. We call this method *channel sampling* and evaluate it in Chapter 5.

Another detection method uses an estimate of channel quality to infer congestion. In this scheme, each node maintains link statistics and signals congestion anytime loss rates rise above a given threshold. This mechanism is similar to the way in which TCP uses loss as a signal for its congestion control mechanisms. Additionally, it may be beneficial to combine all or some of these congestion detection approaches in an effort to detect congestion as early as possible. We leave such an analysis for future work.

Congestion mitigation is the mechanism by which nodes in a given radio neighborhood throttle their transmissions to prevent queues at their next-hop node from overflowing. When a node overhears a packet from its parent with the congestion bit set, it stops forwarding data, allowing the parent to drain its queues. Without such a feedback mechanism, packet buffers could easily be overrun when a wave of traffic flows through the network. If a path experiences persistent congestion, hop-by-hop backpressure will eventually reach the source, allowing application-level flow control (described later in Section 4.6), to throttle the source rate. Figure 4-1 shows a typical scenario where hop-by-hop flow control would be employed.

Relaying a node's congestion state can be problematic when backpressure needs to propagate through multiple hops. For example, when a parent sets its congestion bit, its children stop transmitting, thereby preventing themselves from informing their own children when they become congested. We solve this problem by allowing a congested node to send out one additional packet after it receives backpressure from its parent. A congested node may also send one additional packet per received packet, to compensate for children not hearing a packet that indicates congestion. Each unexpected packet—those received after a node sets its congestion bit—is a signal that congestion state has not propagated to all children. This policy does risk overflowing a parent's output queue by one packet, but does so in order to prevent the transmission

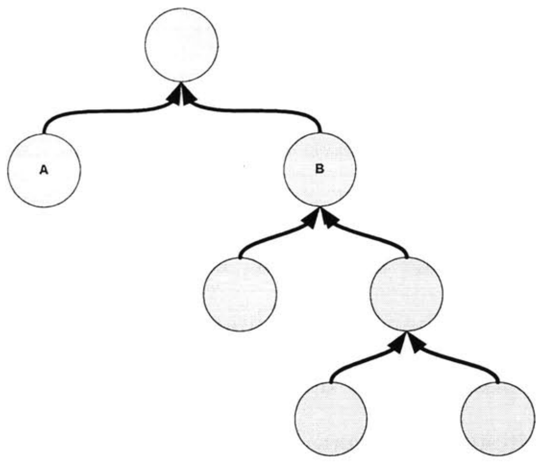

Figure 4-2: Unbalanced network topologies make delivering a node's fair share difficult. In the diagram above, node B should get 5x the bandwidth of node A, since it provides transit for many nodes. In a congested environment, a traditional CSMA MAC would give nodes A and B equal access to the channel, leading to a great degree of unfairness. The source rate limiting scheme described in Section 4.3 aims to reduce the unfairness in these and other topologies.

of a potentially larger set of packets from children who did not hear the original backpressure signal.

4.3 Rate limiting

Congestion can occur anywhere in the network because of the variability of channel conditions and workloads. These points of congestion usually result in an increase in the noise floor accompanied by a precipitous drop in the packet delivery rate. As network diameter grows, it becomes particularly problematic if transit traffic is dropped due to congestion, because the network has already expended a significant amount of energy and bandwidth transmitting the packet over many hops (a problem referred to as *livelock*).

Moreover, there is a natural tendency for the network to deliver traffic originating close to a sink at the expense of traffic sourced deeper inside the network. Figure 4-2 shows a topology that could arise in practice and result in a high degree of unfairness in a congested network. Because of the shared nature of the wireless spectrum and how the CSMA MAC layer operates, nodes A and B would each receive access to half of the available bandwidth. This situation is undesirable because node B must provide transit for traffic originating from five nodes while node A must only do so for one. Ideally node B should receive five times the bandwidth of node A if each node in the network has data to send. Consequently, to address the problems caused by unfairness and the rise in the noise floor, we propose using *source rate limiting.*

The source rate limiting scheme we evaluate works as follows. Note that we make the simplifying assumption that all nodes offer the same traffic load and that the routing tree is not significantly skewed. A more general approach that better handles variable rates would require nodes to propagate their rates. For simplicity, we use a passive approach that relies on nodes monitoring transit traffic to determine the rates at which they should source data. In this approach, each node listens to the traffic its parent forwards to estimate N, the total number of unique sources forwarding data through the parent. Each node uses a token bucket scheme to regulate the rate at which *locally-generated* traffic is admitted into the network. A node accumulates one token every time it hears its parent forward N packets, up to a maximum number of tokens. The node only admits packets from its local applications when its token count is above zero, with each send costing one token. This approach rate limits the node to source traffic at the same rate as each of its siblings. Transit traffic at a node need not be rate limited because it was previously admitted to the network using our source rate limiting policy. Should this policy be inadequate, hop-by-hop backpressure will ensure that distant sources reduce their transmission should a "hotspot" develop.

We evaluate this simple source rate limiting scheme in Chapter 5 both in isolation and in concert with the other congestion control mechanisms.

4.4 The MAC layer

Although nodes can react to congestion using the above network-layer mechanisms, they cannot always react to congestion fast enough to prevent buffer losses without help from the MAC layer. A carrier sense multiple access (CSMA) MAC can aid congestion control.

A standard CSMA MAC layer gives all nodes competing to transmit an equal chance of success. However, during times of congestion, this approach can lead to reduced performance due to a congested node's inability to quickly propagate congestion control feedback to its neighbors. For example, consider a high fan-in scenario, where several nodes are forwarding data through a common parent. On average, the parent node will gain access to the channel only after half of its neighbors have transmitted. However, because the parent is congested, it may not have enough buffer space available to store packets from half of its children. Hence, the parent has no choice but to drop packets that some of its children forward to it. Consequently, it is imperative that congested nodes have *prioritized access* to the wireless medium.

In order to address this issue, we adopt a technique that Aad and Castelluccia advocate [7], making the length of each node's randomized backoff (before every transmit cycle) a function of its local congestion state. If a node is congested, its backoff window is one-fourth the size of a non-congested node's backoff window, making it more likely that a congested node will win the contention period, allowing queues to drain. This approach also increases the likelihood that hop-by-hop congestion control information (whether or not queues are backlogged) will reach a node's neighborhood.

4.5 The hidden terminal problem

As described in Chapter 3, hidden terminals occur when two senders that are not in radio range transmit to a common receiver. One way of reducing collisions between hidden terminals is to exchange RTS/CTS control packets before communicating. Although these control packets can collide, and some non-colliding transmissions may be prevented, the RTS/CTS exchange eliminates most data packet collisions. The added cost of the RTS/CTS exchange is worthwhile when data packets are substantially larger than control packets. However, in sensor networks, data packets are usually small [21], and on some platforms the RTS/CTS exchange would incur a 40% overhead [54]. Consequently, we do not evaluate this mechanism.

Woo and Culler propose a simple strategy to alleviate hidden terminals in tree-based topologies [54]. When a node overhears its parent finish sending a packet, it waits for one packet-time plus a guard time, to avoid a likely hidden terminal collision with its grandparent (Figure 3-3). We evaluate this *delay* strategy with our other congestion control strategies in Chapter 5.

4.6 Application adaptation

Application-level mechanisms play an important role in preventing congestion. In our TinyOS implementation, when the networking stack is not ready to accept additional data, it signals applications via send failures. It is then up to the application to respond appropriately. Some applications will simply wait until the stack is ready again (the strategy we evaluate). Others may adjust their send rate via an AIMD controller[1] or a similar mechanism. Generally, in resource-constrained wireless sensor networks applications should only allow a small number of packets to be outstanding in the networking stack at any given time. This policy is important on resource constrained nodes using a simple FIFO queue to schedule packet transmissions because locally generated traffic could easily consume the entire output queue, resulting in most transit traffic being dropped. Finding the appropriate balance between transit and locally generated traffic is critical for high end-to-end performance and may argue for some degree of separation between the two classes of traffic, which we leave for future work.

4.7 Evaluation implementation

With the widespread use of TCP on the Internet, congestion control is a service that most developers expect their networking stack to provide. Likewise, in sensor networks the communication model should shield developers from the details of conges-

[1]An additive-increase multiplicative-decrease, or **AIMD**, controller additively increases a parameter when it receives positive feedback and multiplicatively decreases the parameter when it receives negative feedback. TCP uses an AIMD controller to adjust the size of its congestion window and probe for additional bandwidth.

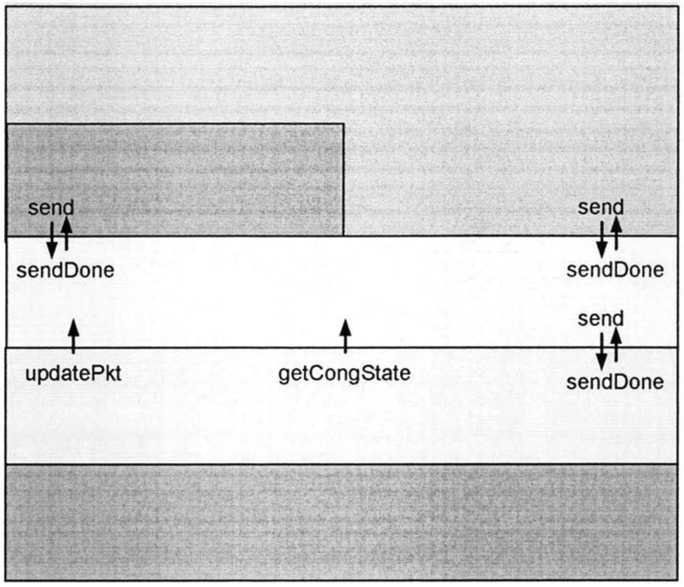

Figure 4-3: The Fusion congestion control suite is implemented in TinyOS as a congestion-aware queue and MAC sitting above the raw radio layers but below the application layer.

tion control, allowing most applications to benefit without developers having explicit knowledge of congestion control subtleties. To that end, we implemented Fusion as a congestion-aware queue that resides between the application and the lower-layer networking protocols. In addition, we made several enhancements to the TinyOS MAC layer; however, these modifications are decoupled from the queue, which can operate correctly in isolation. Figure 4-3 depicts where the Fusion queue fits into the overall network stack structure in TinyOS. Applications enqueue packets using standard TinyOS conventions and receive a signal when the radio stack processes the packet. The flow of packets through the queue is governed by the congestion control algorithms described in the previous sections.

Chapter 5

Experimental Evaluation

In this chapter we evaluate our integrated approach to congestion control, Fusion, in a 55-node indoor wireless sensor network (Section 2.3). We analyze its component mechanisms, presented in Chapter 4, both in isolation and in concert to study their relative contributions to performance. Because traffic patterns in deployed sensor networks can be variable, we base our analysis on results from three realistic workloads: a *periodic workload*, a *high fan-in workload*, and a *correlated-event workload.* The metrics we use to measure network performance—efficiency, imbalance, throughput, fairness, and latency—are those presented in Chapter 3. What follows is a brief summary of the congestion control techniques we evaluate, with the remaining sections providing a detailed discussion of their performance under each workload.

5.1 Congestion control techniques

The results presented in the subsequent sections discuss the performance of the following six congestion control strategies:

1. *Occupancy*: hop-by-hop flow control using queue occupancy to detect congestion (Section 4.2)
2. *Channel sampling*: hop-by-hop flow control using channel sampling to detect congestion (Section 4.2)

Strategy	**Remarks**	**Parameters**
Queue occupancy	Hop-by-hop flow control using queue occupancy (Section 4.2); α indicates the fractional queue occupancy at which congestion is indicated.	Queue size of eight, $\alpha = 0.25$.
Channel sampling	Hop-by-hop flow control using channel sampling (Section 4.2). N indicates the number of epochs of length E that the channel is sensed for; α indicates the EWMA averaging parameter.	$N = 4$, $E = 100$ milliseconds, $\alpha = 0.85$.
Delay	The queue occupancy strategy augmented with the *delay* technique as described in Section 4.5. After overhearing the end of a parent's transmission, we backoff for τ milliseconds, slightly more than one packet-time on the Mica2's CC1000 radio [15].	$\tau = 40$ milliseconds.
Rate limiting	We implement a simple *rate limiting* strategy as described in Section 4.3.	Token bucket has a depth of 10. Active sources discovered over a 4 second window.
Fusion	This strategy simultaneously combines the queue occupancy, forwarding delay, and rate limiting algorithms.	As above.
No congestion control	No congestion control related changes are made to the network layer or MAC. Transmission is attempted as soon as data enters the outgoing queue (after carrier sense and MAC backoff).	None—uses an unmodified B-MAC.

Table 5.1: The congestion control strategies evaluated in this thesis and their relevant parameters.

3. *Delay*: hop-by-hop flow control with hidden terminal avoidance as (Section 4.5)
4. *Rate limiting*: source rate limiting (Section 4.3)
5. *Fusion*: combining delay and rate limiting (Section 4.1)
6. *No congestion control*: an unmodified network stack

We summarize each congestion control strategy in Table 5.1, providing the values of the parameters for each scheme. For every strategy except "no congestion control", we use B-MAC with the prioritization features described in Section 4.4. For the "no congestion control" strategy, we use the unmodified version of B-MAC (Section 2.3.3) included with TinyOS.

5.2 Experimental setup

We evaluate each of our congestion control techniques under three realistic workloads. In the periodic workload, nodes generate readings at a fixed rate, representing a monitoring network. In the high fan-in workload, we increase the average number of children per parent (by constraining the routing topology) to model a tiered or clustered network topology. The correlated-event workload evaluates our strategies under an impulse of traffic, which is common in many tracking and detection applications.

For each experimental data point in the periodic and correlated-event workload results, we report an average over 15 runs to one sink (node 6, see Figure 2-3 for its location). Our high fan-in workload results are the average of 5 runs. The graphs for the periodic and correlated-event workloads contains 99% confidence intervals, and the graphs for the high-fan workload contain 95% confidence intervals (the decrease in confidence is due to fewer runs). We combine runs taken during different times of the day and on different days of the week. The traffic statistics collection phase of each run lasts 240 seconds for the periodic and high fan-in workloads, and 60 seconds for the event experiment. Therefore, the total run-time for each data point is 3600 seconds for the periodic workload, 1200 seconds for the high fan-in workload, and 900 seconds for the correlated-event workload.

As described in Section 2.3.2, nodes cooperatively forward data to the sink using a multi-hop routing tree. Nodes select individual forwarding paths using ETX and propagate routes using DSDV. All packets sent through the wireless sensor network are 36 bytes in length. During the experiments the only other traffic running through the network is from infrequent routing updates, which are sent every ten seconds per node. In addition, we set the radio frequency to reduce the amount of outside interference from other wireless sensor networks in the building. We allow routes to stabilize for 120 seconds prior to the data collection phase of our experiments, and we pin routes for the duration of the experiment once the stabilization phase completes. Pinning routes ensures that the routing protocol does not influence the outcome of our experiments.

We evaluate all metrics as a function of per-node offered load, which ranges from 0.25 packets/s to 4 packets/s for periodic and high fan-in workloads. For the correlated-event workload, the event size ranges from 1 to 8 packets. Since the link-level throughput for the Mica2 caps at approximately 40 packets/s and our network has 55 nodes, we can be sure that our network becomes congested at 4 packets/s. In addition, all packets are transmitted reliably with up to three retransmissions per-link.

5.3 Periodic workload

The *periodic workload* models a typical monitoring sensor network in which sensor nodes generate readings at fixed time intervals. Deployments exhibiting this traffic pattern are quite common in practice [22, 33, 49]. In this workload, each node sources traffic at some fixed rate and helps to forward other nodes' traffic to a sink. To avoid synchronizing periodic reports from different nodes, we introduce a random jitter, which is small relative to the report rate, at the beginning of the experiment. Figure 5-1 shows a snapshot of the routing topology used to forward packets toward the sink.

We note here that it is deceptively difficult to provision a wireless sensor network to obviate the need for an congestion control, even under this simple workload.

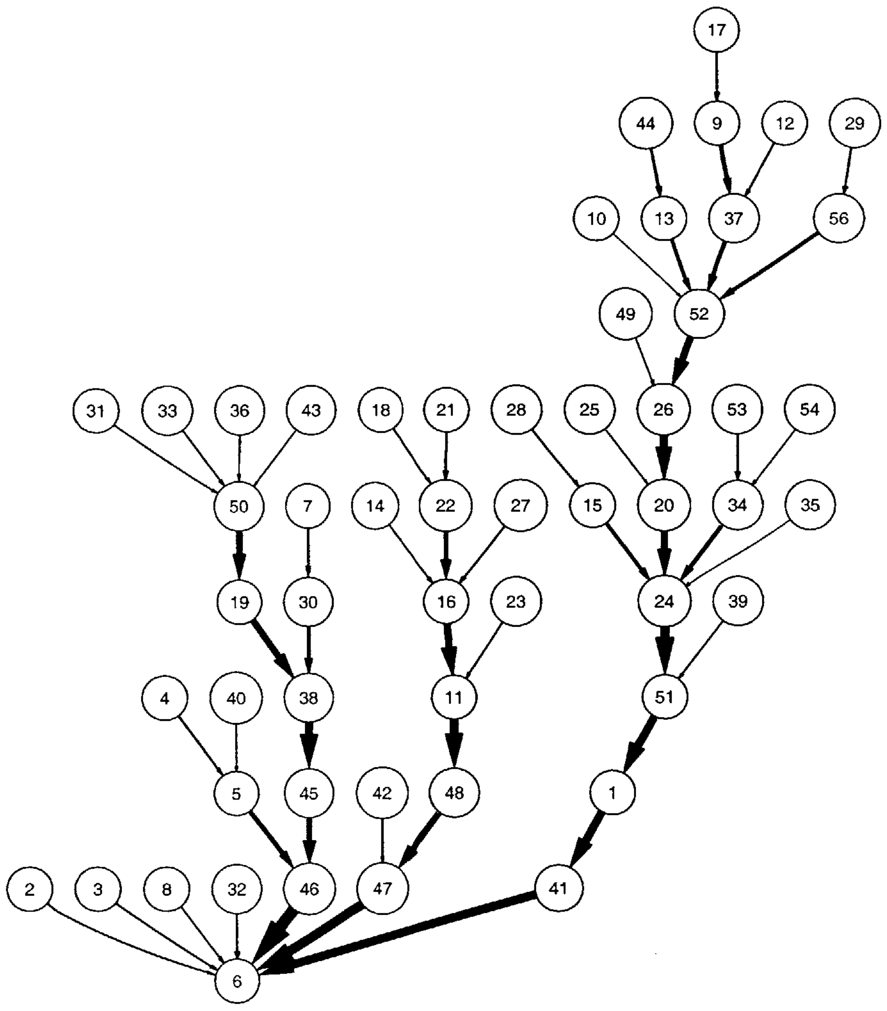

Figure 5-1: The routing topology in one run of an experiment with Fusion flow control and each node offering 1 packet/s, as formed by the ETX (Section 2.3.2) path selection metric. The thickness of each edge is proportional to the number of packets the node at the head of the edge received from the node at the tail of the edge.

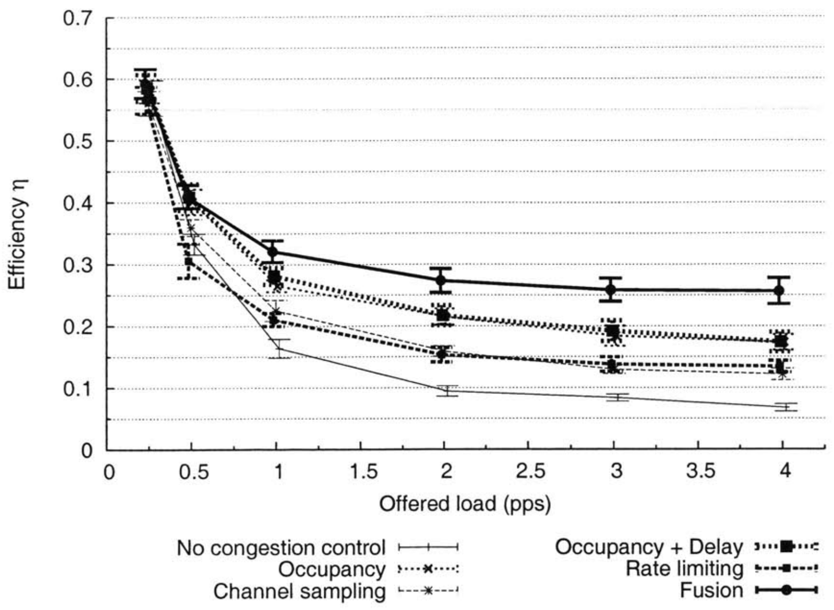

Figure 5-2: Average network efficiency η versus per-node offered load under a periodic workload. 99% confidence intervals are shown.

Even though a network designer knows the aggregate offered load *a priori*, the unpredictable nature of radio transmissions makes it is almost impossible to know the actual forwarding topology of the network, making fine-grained channel bandwidth allocation difficult. In addition, the forwarding topology changes with time in response to changing network conditions.

5.3.1 Periodic workload: Network efficiency

As described in Section 3.4, network efficiency measures the fraction of transmissions that go towards getting packets to an access point. Specifically, efficiency (defined in Equation 3.1) is the ratio of the number of packets received at the sink, weighted by the distance each packet travels in hops, to the total number of transmissions, including retransmissions and dropped packets. Figure 5-2 shows how network efficiency η varies with per-node offered load for each congestion control strategy. First, note the decreasing trend for each congestion control strategy. This trend is expected, because as the number of transmissions increases, the noise floor of the network rises, increasing the probability of packet corruption and retransmission. Additionally, the probability of collisions due to MAC contention and hidden terminals increases. These factors drive η down as offered load grows.

Rate limiting offers an incremental improvement in efficiency for the same reason that the efficiency vs. offered load trend is downward: fewer packet transmissions. In particular, rate limiting reduces the offered load near the sink of the network, where congestion and contention are worst. However, as the offered load increases, hidden terminals and interference prevent this strategy from exceeding the efficiencies of the other strategies.

Hop-by-hop flow control offers an additional improvement in efficiency, but succeeds for a different reason than rate limiting. Rather than reducing contention between the transit and leaf nodes of the network, hop-by-hop flow control improves efficiency by throttling transmissions based on queue contention in the local neighborhood. In addition, as offered load increases, queue occupancy congestion detection consistently outperforms channel sampling. This suggests that queue occupancy is at least as good as channel sampling as an indicator of congestion.

Combining these two flow control strategies in the Fusion scheme yields the highest gains in efficiency. Hop-by-hop flow control helps to throttle transmissions at every link in the network, while the rate limiting mechanism meters traffic being admitted into the network. Moreover, these strategies are complementary because rate limiting delays transmissions even when hop-by-hop flow control would allow them. Together, these two techniques achieve high levels of efficiency *even after* the network reaches saturation, increasing efficiency by a factor of three. However, although these gains are substantial, efficiency is still quite low, barely reaching 0.3 at moderate offered loads. With two out of three transmissions being wasted, there is still room for improvement.

5.3.2 Periodic workload: Imbalance

Recall from Section 3.4 that imbalance, or ζ, is a measure of how well a node forwards data. We define imbalance as the ratio of the number of packets received at a node to the number of packets that were received at its parent. This metric is particularly

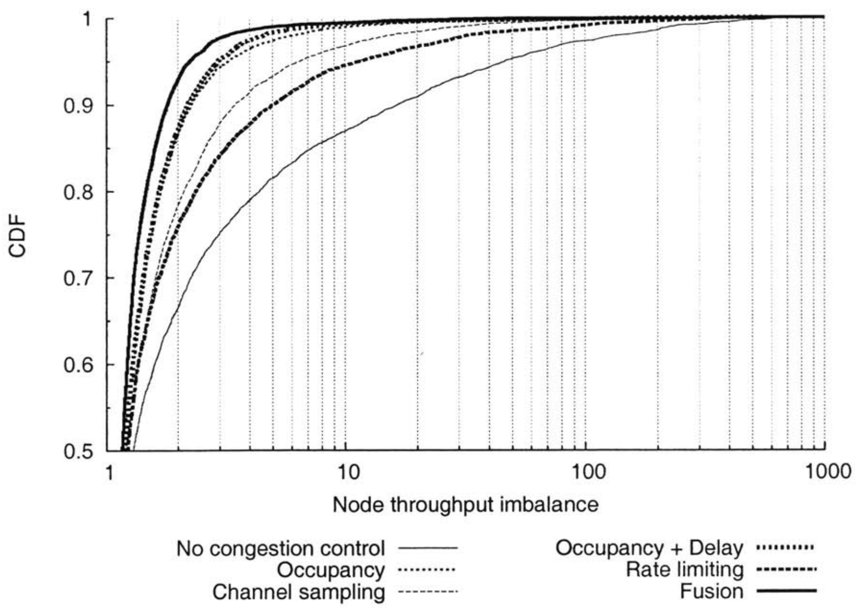

Figure 5-3: Node imbalance ζ (CDF) for different flow control strategies. Each node offers 4 packets/s.

useful for identifying hotspots (nodes with high ζ) in a network that account for a large fraction of the wasted transmissions.

With an offered load of 4 packets/s, we plot distributions of node imbalance ζ for different congestion control strategies in Figure 5-3. These results summarize node imbalances over multiple runs of the periodic experiment.

Without any congestion control strategy, the network has many hotspots: approximately five nodes (the 90th percentile) have an imbalance greater than 50—meaning five nodes received 50 times as much traffic as they could forward. Furthermore, the tail of the imbalance CDF without congestion control is very heavy, indicating the presence of hotspots in the network that are not successfully forwarding any traffic they receive. Rate limiting also exhibits a heavy tail, indicating that rate limiting does little to help these nodes.

Channel sampling and occupancy-based congestion control both remove hotspots from the network. We see a marked synergistic improvement when we combine the congestion control strategies in the Fusion scheme.

Figures 5-4 and 5-5 qualitatively show the amount of traffic received by each node in one experimental run. The figures show a periodic workload of 4 packets/s using

Fusion and no congestion control strategies, respectively. The thickness of each link is proportional to the number of packets the node at the head of the edge receives from the node at the tail of the edge.

In Figure 5-4 (with Fusion), the relatively thick edges in the graph form a *forwarding backbone* over which most traffic is sent. This forwarding backbone shows that nodes are correctly funneling traffic towards the sink at rates that can be sustained. The rate limiter helps to shape traffic such that leaf nodes, especially those near the sink, do not overload the network. In addition, a "conservation of packets" property holds at most nodes: for nodes that route traffic, several thin incoming links usually result in a much thicker outgoing link. These nodes have an imbalance close to 1.

In contrast, in Figure 5-5 (with no congestion control) there is no clear forwarding backbone, and the conservation of packets property does not hold at many nodes. For example, one or more thick edges entering a node lead to one thin edge exiting the same node. This implies a large number of buffer or wireless drops, and explains the hotspots we see in Figure 5-3. Also note the lack of thick edges in the densely-populated upper-left hand corner of the map. This suggests a large number of wireless collisions in that geographic region.

5.3.3 Periodic workload: Throughput and fairness

Next, we measure the aggregate received throughput at the sink, without regard to which nodes deliver the data. Figure 5-6 shows that as offered load increases, using the non-rate limiting congestion control strategies—occupancy, channel sampling, and delay—result in the highest throughput. This follows because rate limiting reduces transmission rates to allow nodes deep in the network to reach the sink.

The throughput trend is of secondary importance, however, since fairness decreases substantially without congestion control. Figure 5-7 shows the distribution of throughput that the sink receives from each node at an offered load of 2 packets/s. Note that without congestion control, more than 40% of the nodes deliver less than 1 packet every 100 seconds, making that part of the network essentially useless. While

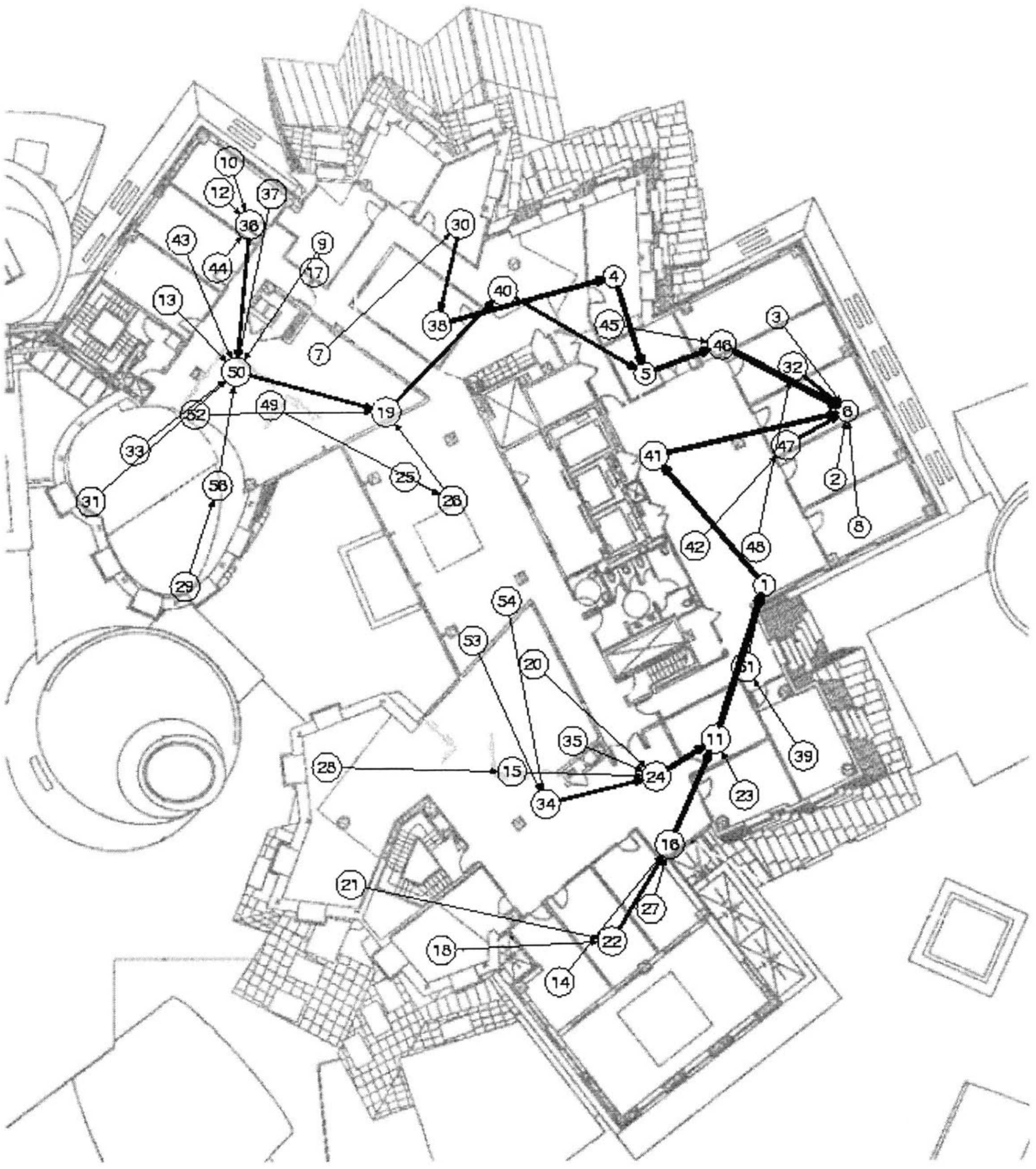

Figure 5-4: Traffic flow in one run of a *Fusion* congestion controlled experiment with each node offering 4 packets/s. The thickness of each edge is proportional to the number of packets the node at the head of the edge receives from the node at the tail of the edge. The deployment contains 55 nodes spread out over 16,076 square feet and node 6 serves as the network sink.

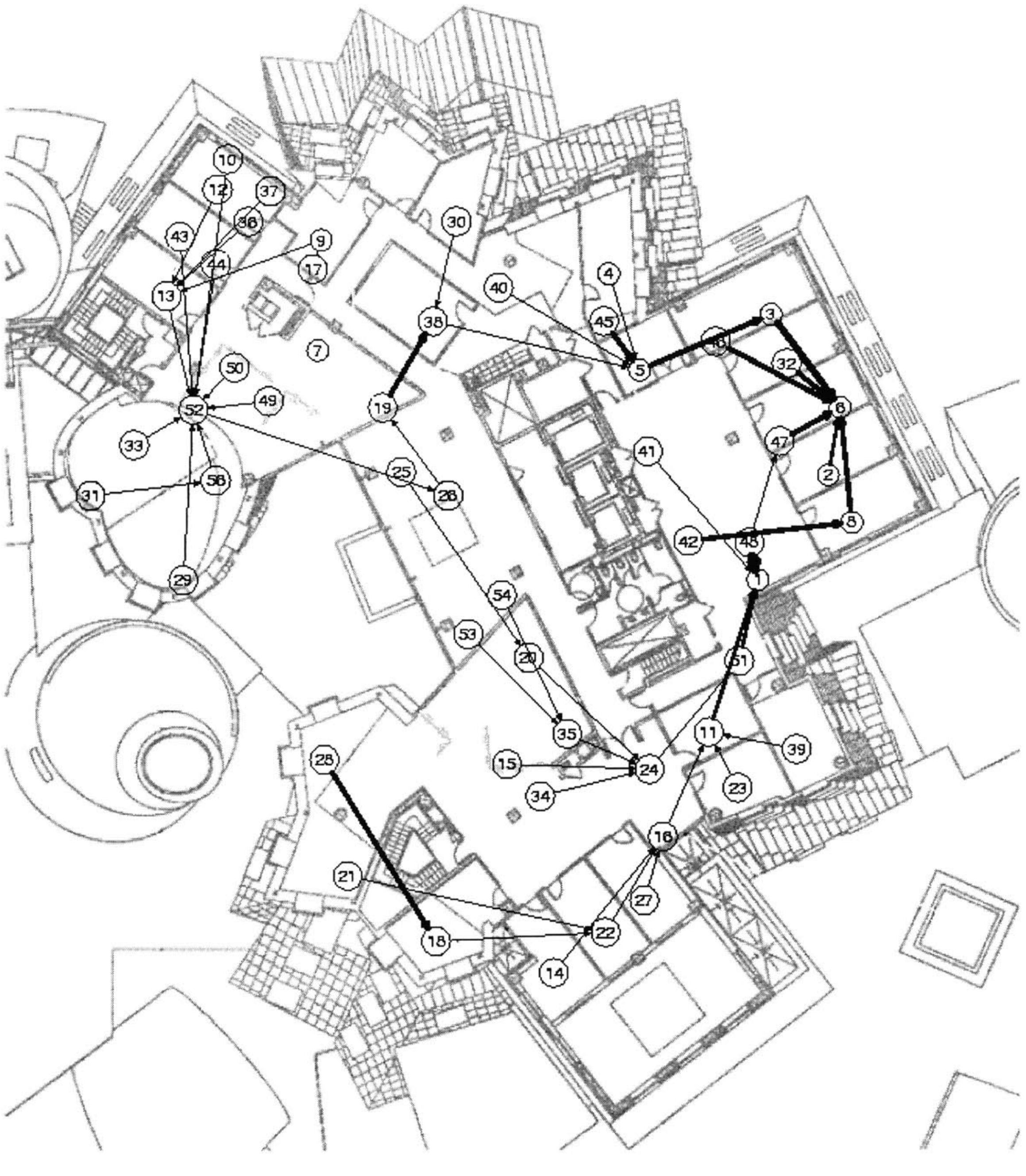

Figure 5-5: Traffic flow in one run of an experiment with *no congestion control* and each node offering 4 packets/s. The thickness of each edge is proportional to the number of packets the node at the head of the edge receives from the node at the tail of the edge. The deployment contains 55 nodes spread out over 16,076 square feet and node 6 serves as the network sink.

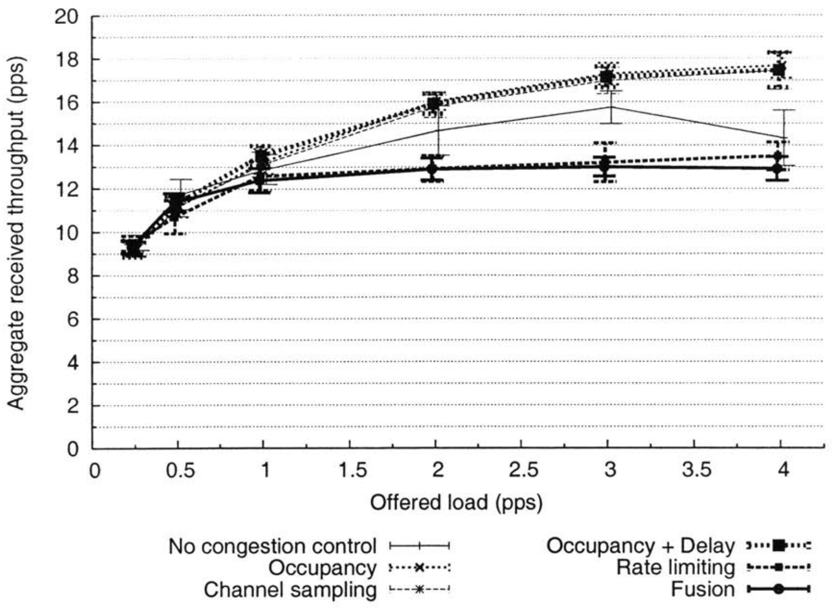

Figure 5-6: Average aggregate received throughput (measured at the sink) versus per-sensor offered load under a periodic workload. 99% confidence intervals are shown.

congestion control sacrifices the high throughput of a minority of nodes, it distributes bandwidth between nodes more fairly.

Table 5.2 shows the data reporting periods that any given percentage of nodes in the network can achieve when every sensor offers 4 packets/s. In the second column, we see that without any congestion control, no throughput guarantees can be made for 25% of the nodes. In contrast, nodes using an occupancy-based hop-by-hop congestion control strategy can cover 90% of the network. If, however, we are only interested in the throughput 10% of the nodes in the network can achieve, no congestion control is the best strategy. This regime, however, is unlikely to be of much interest to sensor network designers, because 10% of the nodes would provide poor network coverage, particularly because these nodes are the ones closest to the sink.

As we vary offered load between 0.25 and 4 packets/s, we see the same trends for aggregate throughput. Without any congestion control, aggregate throughput increases as show in Figure 5-6. However, the network mostly delivers data from nodes one hop away from the sink, resulting in a decrease in fairness as shown in the right hand side of Figure 5-8. Without source rate limiting, congestion control mechanisms suffer a similar fate, because nodes in the core of the network have more

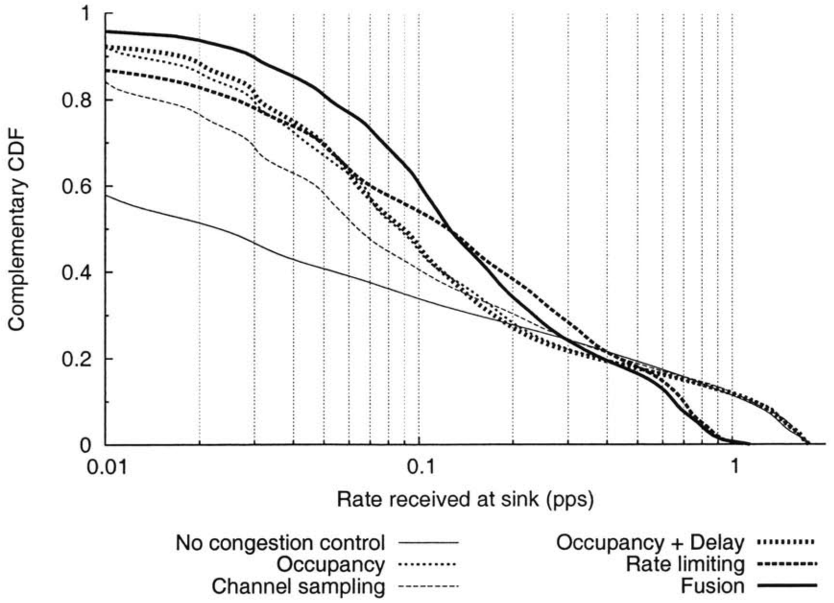

Figure 5-7: Per-node throughput received at the SAP (complementary CDF) under a periodic workload. Each node offers 2 packets/s.

	Maximum period					
Coverage	No cong. ctrl	Occupancy	Channel sampling	Occ. + Delay	Rate limiting	**Fusion**
100%	∞	∞	∞	∞	∞	∞
95	∞	108 s	∞	123 s	∞	**43 s**
90	∞	62 s	100 s	71 s	250 s	**35 s**
75	∞	29 s	38 s	33 s	31 s	**15 s**
50	38 s	12 s	13 s	12 s	8.5 s	**7.2 s**
25	4.2 s	3.4 s	3.8 s	3.2 s	2.9 s	**3.0 s**
10	840 ms	870 ms	980 ms	950 ms	1.4 s	**1.5 s**

Table 5.2: Network coverage broken down by congestion control strategy. Given that a network designer wants the network coverage shown in the left hand column, that percentage of nodes operating under different congestion control strategies can achieve at least the report rate show in the rightmost columns. The offered load in the network is periodic at 2 packets/s per node.

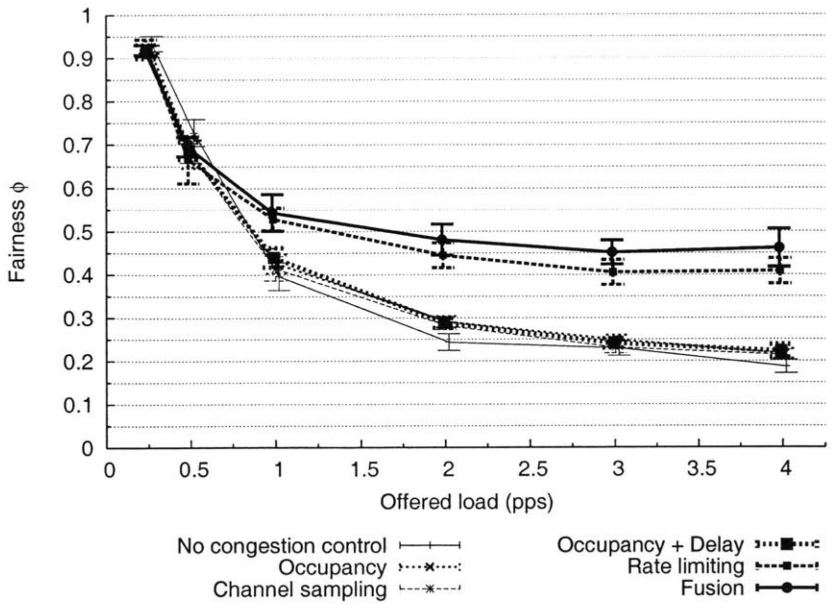

Figure 5-8: Average fairness ϕ versus per-node offered load under a periodic workload. 99% confidence intervals are shown.

opportunities to inject packets into the network, because they have fewer hops over which hop-by-hop backpressure can act. Rate limiting dramatically improves fairness because space becomes available in node transmission queues for traffic from the edges of the network.

Figure 5-9 shows median node throughput as a function of per-node offered load. Below an offered load of 0.5 packets/s, the network is in an underloaded state, and median received throughput increases with offered load. Above 1 packet/s, nodes need a congestion control strategy if more than 50% of the nodes are to provide any traffic at all. This result quantifies the high degree of unfairness that nodes at the edges experience when the network is an a state of congestion collapse. At least half the nodes running the Fusion strategy are able to maintain at least 0.1 packets/s as offered load increases because rate limiting prevents the core of the network from overwhelming the edges.

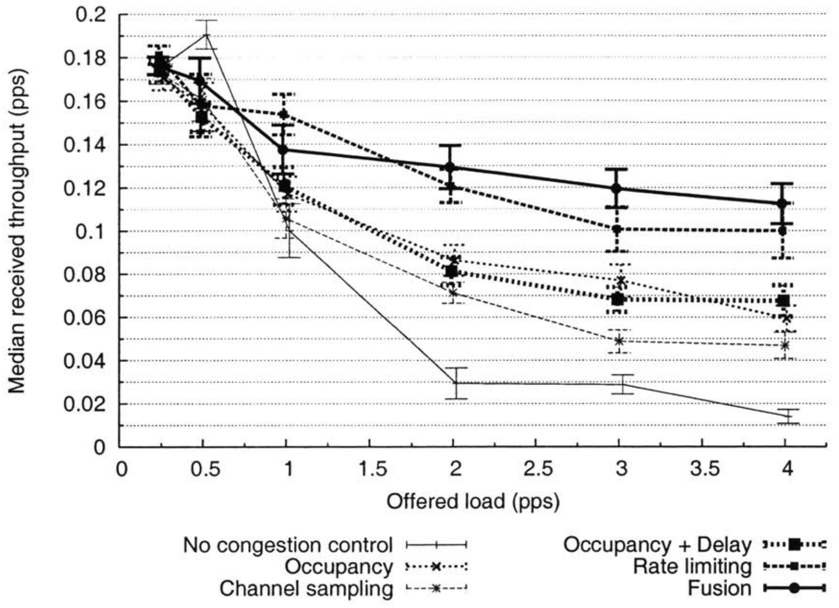

Figure 5-9: Median throughput as a function of offered load under a periodic workload. 99% confidence intervals are shown.

5.3.4 Periodic workload: Latency

Figure 5-10 shows how the median packet latency varies with offered load. We measure latency from the application-level transmit time on the originating node to the time at which the sink receives the packet. Since we only measure the latency of packets received at the sink, this metric must be viewed with fairness in mind. The primary source of latency in the network is queuing delay, which increases as the number of hops a packet must traverse grows. As we increase offered load from 0 to 0.5 packets/s, queues start to build and latency increases. Beyond an offered load of 1 packet/s for all strategies except Fusion, fairness declines sharply because the packets from edge nodes are dropped. Thus, latency decreases with increasing offered load because the nodes that get through are in the core of the network, closer to the sink. Since the Fusion strategy is the fairest, as offered load increases, a greater proportion of the packets received are from the edges (many hops away), and consequently, latency is higher.

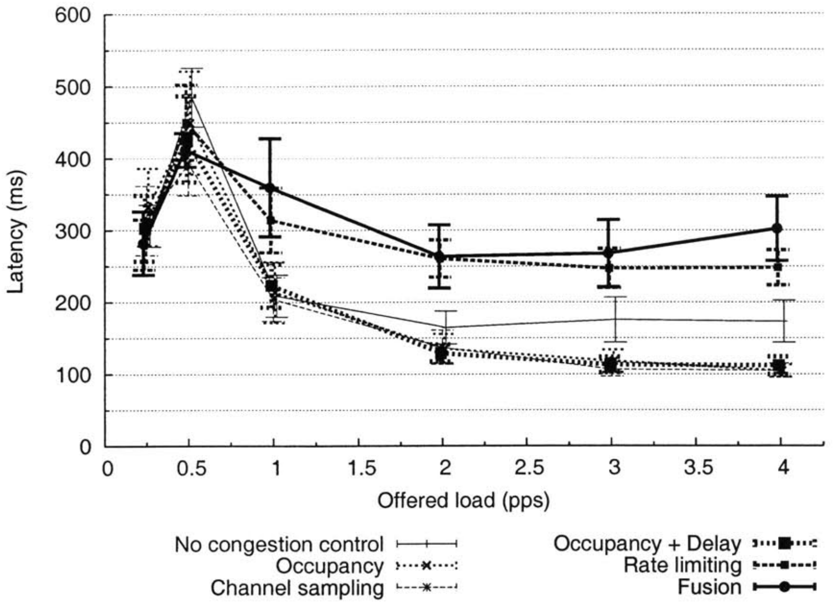

Figure 5-10: Median packet latency as a function of offered load under a periodic workload. 95% confidence intervals are shown.

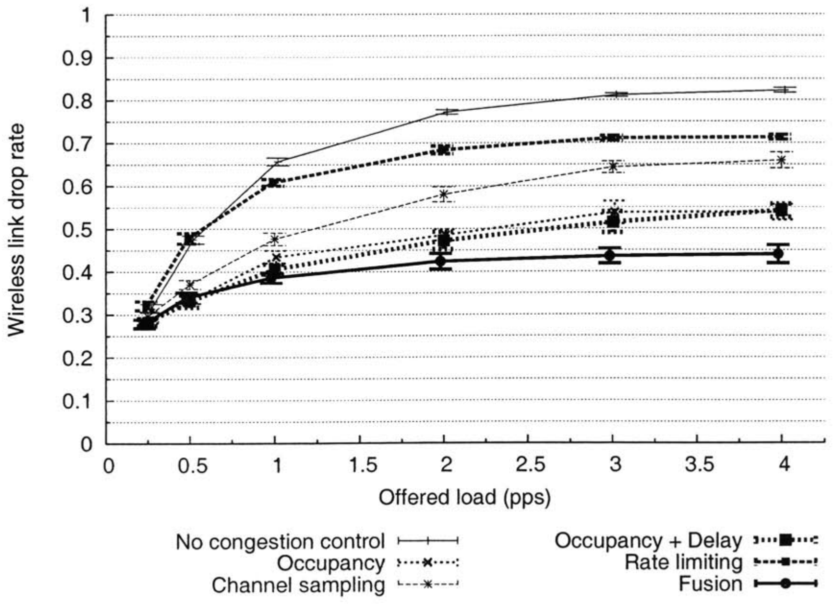

Figure 5-11: Average network-wide wireless link channel rate (before retransmissions) versus per-node offered load under a periodic workload. 99% confidence intervals are shown.

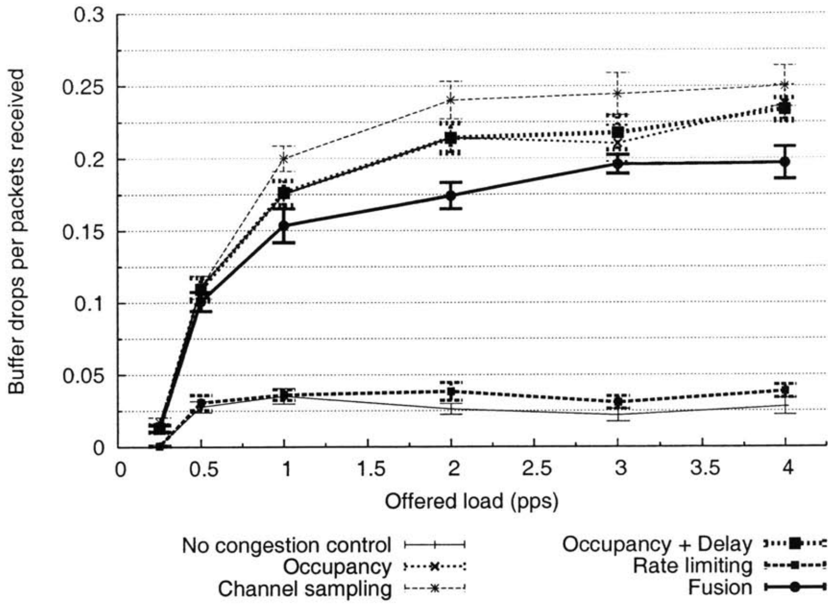

Figure 5-12: Total network-wide buffer drops per network-wide received packets as a function of per-node offered load under a periodic workload. 99% confidence intervals are shown.

5.3.5 Periodic workload: Sources of loss

Figure 5-11 shows network-wide wireless channel loss rates as a function of offered load. We calculate loss rate by dividing the sum transmission count (including retransmissions) by the sum reception count. As expected, the wireless loss rate increases for all strategies as the offered load increases. This trend is caused by rise in the noise floor, and possibly an increase in the number of collisions due to hidden terminals.

The "no congestion control" strategy suffers from the highest loss rates, which approach more than 80% at 4 packets/s. Channel sampling and rate limiting occupy the middle ground, with loss rates approaching 60% and 70%, respectively. The occupancy-based congestion control strategies perform even better, keeping loss rates under about 50%. The Fusion scheme performs the best overall, with close to a 50% decrease in drops at 4 packets/s. Surprisingly, there is no statistical difference in wireless drop rate between the occupancy + delay strategy and the occupancy strategy. This observation suggests that either hidden terminals are not the primary problem in our network, or that the delay strategy does not often avoid them.

Figure 5-12 shows total network-wide buffer drops per packets received, as a function of offered load. This figure measures the probability that a packet will be dropped due to buffer overflow, given that it was successfully received. Surprisingly, the buffer drop rate is significantly higher for strategies that include hop-by-hop flow control. This trend is the result of the substantially higher wireless loss rates of the strategies that do not include hop-by-hop flow control. With these latter strategies (no congestion control and rate limiting), nodes do not receive enough packets to fill up their forwarding queues and trigger buffer drops.

5.3.6 Periodic workload: Energy considerations

Performance under a power-saving workload is important to low-power monitoring applications where a large number of nodes periodically sleep (forgetting their congestion state) in between low-rate packet transmissions. There are a wide variety of power-saving strategies proposed for sensor networks. CSMA-based sleep-wakeup algorithms [14, 56] elect a dynamic "backbone" of nodes that forwards traffic for the other power-saving nodes. Other proposals synchronize nodes' sleep-wakeup cycles [58, 50]. In TDMA-based sensor networks, data sources can sleep in all inactive time slots. For any of the above types of networks, the key challenge is designing and evaluating congestion control algorithms that can function soon after the node wakes up.

The congestion control techniques presented in this thesis rely on overhearing the wireless channel to propagate congestion state. When combined with power-saving, it is not immediately clear how the reduction in overhearing time resulting from various sleep intervals will impact congestion control performance. To measure this impact, we constructed a topology and implemented an algorithm that simulates the effects of the sleep-wakeup cycle. In this power-saving workload, a core set of nodes is selected to forward traffic. The remaining leaf nodes send traffic at a low data rate (1 packet/s), sleeping for a given fraction of the period. After waking up, nodes forget their congestion state, listen for the remaining fraction of the period, then send.

Our preliminary results for varying listen/sleep intervals indicate that sleep-wakeup power-saving strategies have a statistically insignificant impact on network performance. In particular, efficiency is dependent on offered rate, but not listen period. Unlike many protocols that rely on overhearing, hop-by-hop flow control is not cumulative: nodes are only concerned with the most recently propagated congestion state. Additionally, in large multi-hop networks, transit nodes—which need to be awake continuously to forward traffic, and hence, fully benefit from congestion control—represent the majority of transmissions in the network. However, we leave a detailed analysis of the effects of power saving on congestion control for future work.

5.4 High fan-in network experiments

Our high fan-in experiments evaluate congestion control in a wireless network where mitigating congestion is very difficult. We choose only a small subset of the nodes to perform the task of routing traffic in the network. Nodes still choose routes using the ETX metric, but only ten nodes out of 55 advertise routes to the sink. Figure 5-13 shows a representative routing topology used in one run of our experiments. Non-uniform node deployments—where a high degree of fan-in is expected—motivate this workload. Heterogeneous node capabilities, such as differences in processor, memory, and power ability, can also result in such a topology.

Note that compared to the topology formed in Figure 5-1 under the periodic workload, this topology has a higher degree of fan-in and a smaller network diameter. The high fan-in makes hop-by-hop congestion control more difficult, since each child must each receive congestion feedback before the aggregate demand on a parent decreases.

5.4.1 High fan-in network: Network efficiency

Figure 5-14 shows network efficiency versus per-node offered load as offered load ranges from 0.25 to 4 packets/s. Comparing Figure 5-14 with Figure 5-2 (network efficiency under the periodic workload), we make the following observations.

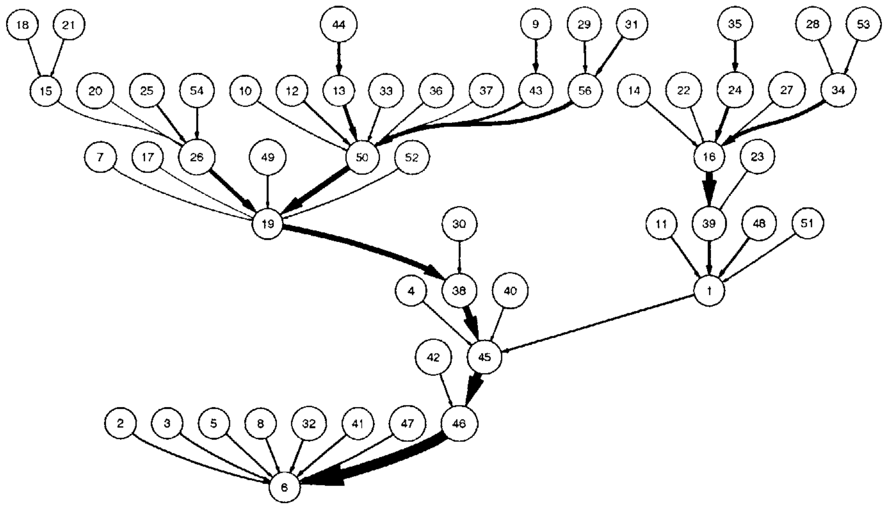
Figure 5-13: Routing topology in one run of a high fan-in experiment with Fusion congestion control and each node offering 1 packet/s, as formed by the ETX path selection metric, restricted to gateway nodes. The thickness of each edge is proportional to the number of packets the node at the head of the edge received from the node at the tail of the edge.

Even at low offered load, network efficiency in the high fan-in network is lower than network efficiency in the periodic network. It is unlikely that poor link selection in the high fan-in network causes the lowered efficiency, because the wireless link drop rates for both the high fan-in and normal networks are equal at low transmission rates. The likely explanation, therefore, is wireless contention and network congestion at high fan-in nodes.

As offered load increases, the trend continues, with efficiency in the fan-in network marginally lower than in the normal network. Note that in a high fan-in topology, congestion control techniques work together to increase performance. Fusion clearly outperforms all strategies as offered load increases.

5.5 Correlated-event workload

The *correlated-event workload* captures the effect of spatially-correlated events in a sensor network. Detection and tracking applications often use an event-oriented reporting model rather than sending a periodic stream of readings. In particular, this

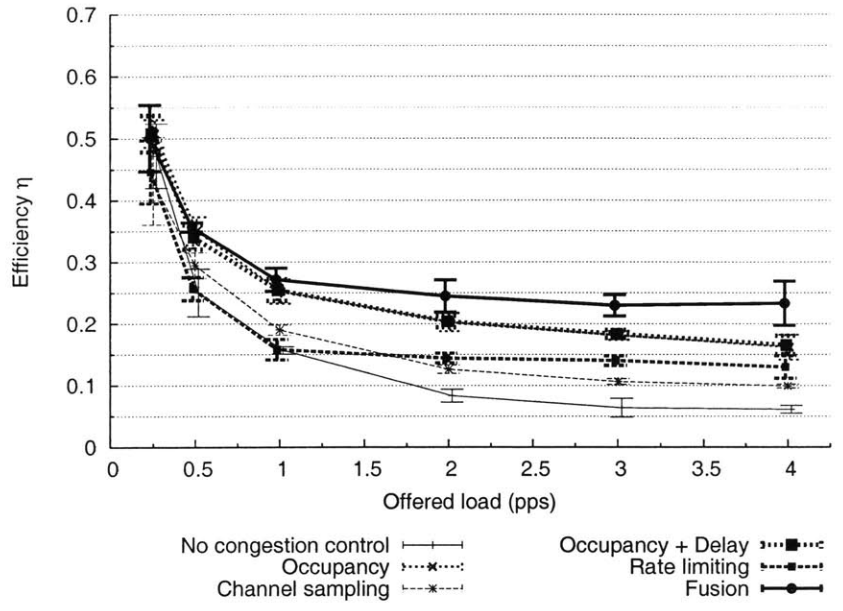

Figure 5-14: Average network efficiency η versus per-node offered load in a high fan-in network. 95% confidence intervals are shown. The curves for Occupancy and Occupancy + Delay overlap.

workload models the effects of a single, synchronized impulse of traffic on network performance.

Instead of collecting analog data from each node in our testbed, we model an event-based workload using the following method. At the beginning of the experiment, we run a distributed level-aware time synchronization protocol similar to the Timesync Protocol for Sensor Networks (TPSN) [19]. Using an Ethernet backchannel, we verified synchronization of 91% of the testbed to within 29 ms (approximately one packet-time) and all 55 nodes in our testbed to within 56 ms (approximately two packet-times).

After the synchronization phase, all nodes in the testbed simultaneously send B packets back-to-back (or, if the channel is carrier-sensed busy, as fast as possible) at the scheduled time of each event. There is sufficient time (20 seconds) between each event to let traffic drain completely from the network.

We evaluate this correlated-event workload by varying the traffic burst size B. It is important to note that for this workload, which consists of a single impulse of traffic, source rate limiting has no impact. Our rate limiting algorithm is designed to operate when there is significant per-node traffic present in the network. Our events do not last long enough for our rate limiter to build up an estimate of sustainable traffic rates. As a result, we omit the rate limiting and Fusion schemes from our data sets for clarity and investigate in detail performance of the hop-by-hop variants.

5.5.1 Correlated-event workload: Network efficiency

Even at small event sizes, the strong time correlation between events necessitates some kind of congestion control. This trend stands in contrast to that of the periodic and high fan-in workloads in which the benefits of congestion control are mainly seen at higher offered loads. Figure 5-15 shows network efficiency as each node's traffic event size B varies between 1 and 8 packets. As expected, we observe that a downward trend still exists for all strategies as network load increases. However, even when $B = 1$, the occupancy hop-by-hop congestion control strategy yields close to an 80% gain in network efficiency over no congestion control. As B increases, the relative improvement in network efficiency (versus the baseline of no congestion control) increases for all strategies. When $B = 8$, the best strategy is approximately two times better than the baseline.

There is a clear benefit to using hop-by-hop flow control for a correlated-event workload to combat congestion caused by the wave of data flowing from the leaves to the sink.

Buffer occupancy augmented with a forwarding delay for hidden terminal reduction (occupancy + delay) performs a little better than just occupancy-based hop-by-hop congestion control when B is small. Without the delay strategy, synchronization in the correlated-event workload makes hidden terminal collisions between a node and its grandparent more likely. The forwarding delay allows a node's grandparent, which normally would exist as a hidden terminal and be prone to collision, a larger window to successfully complete its transmission. At larger values of B, the improvement in network efficiency due to the forwarding delay is negligible. For all event sizes, channel sampling performs noticeably worse than all strategies except for no congestion control.

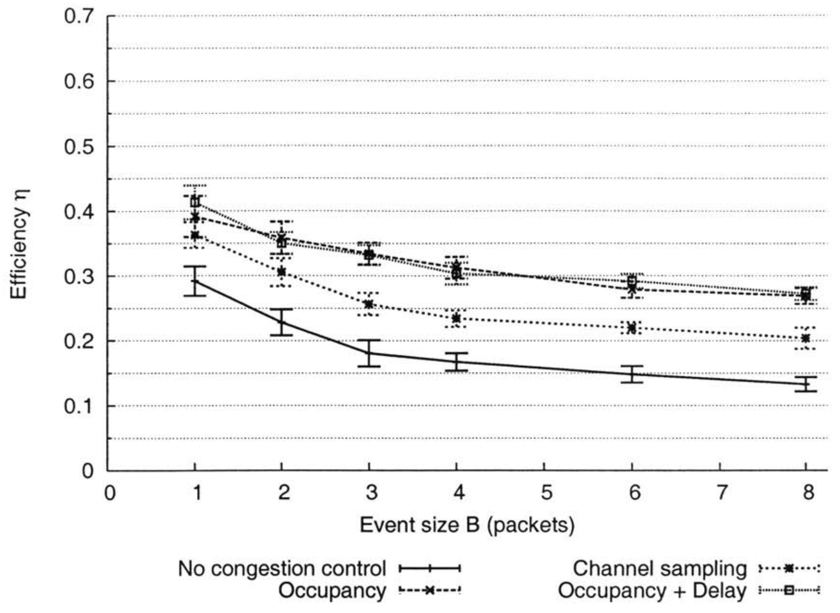

Figure 5-15: Average network efficiency η as a function of event size B under a correlated-event workload. 99% confidence intervals are shown.

5.5.2 Correlated-event workload: Drop rate

For many types of applications designed to detect discrete, non-repeating events, the end-to-end packet drop rate is an important measure of performance. This need for reliability stands in contrast to the periodic workload where often it is reasonable to assume that subsequent reports will supersede any lost data. Figure 5-16 shows the end-to-end drop rate as a function of event size B for various congestion control strategies. Note how all strategies perform better than no congestion control. In some cases, the lack of any congestion control can increase the end-to-end drop rate by almost 35%. Hop-by-hop flow control alleviates congestion in this workload because the backpressure desynchronizes packet transmissions.

5.5.3 Correlated-event workload: Latency

In Figure 5-17 we show how packet latency rises when using any of the congestion control strategies. This increase is to be expected because all congestion control strategies operate by delaying transmissions. By decreasing the rate at which queues drain, wireless contention and collisions are reduced at the cost of increased queuing

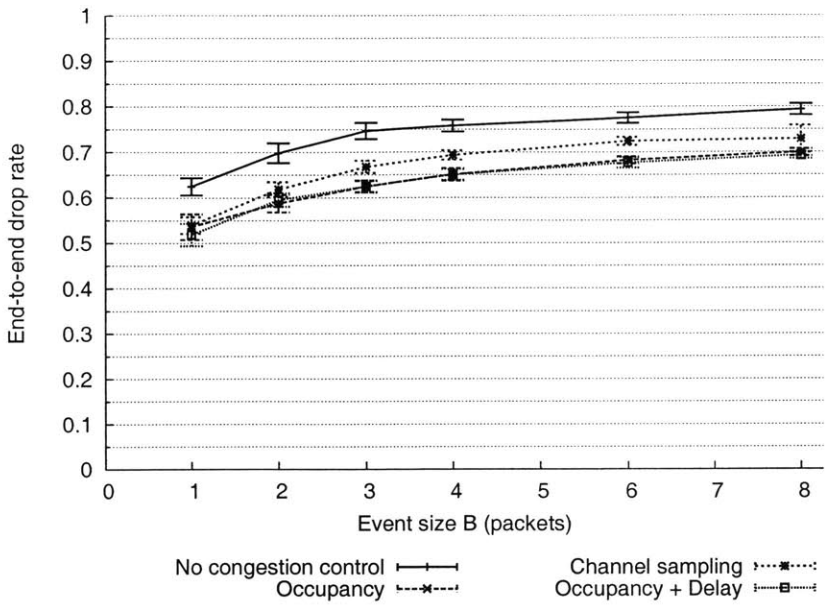

Figure 5-16: End-to-end drop rate as a function of event size B under a correlated-event workload. 99% confidence intervals are shown.

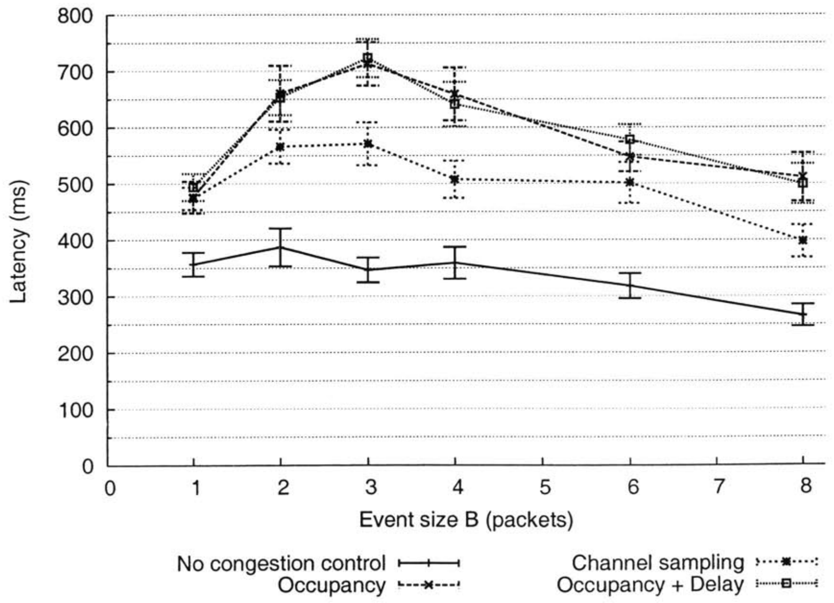

Figure 5-17: Median packet latency as a function of event size B under a correlated-event workload. 95% confidence intervals are shown.

delays. For small event sizes, the increase in latency is in the neighborhood of 40%. However, the maximum increase in latency is seen when the event size is 3, resulting in a latency increase of over 100% when compared to no congestion control.

Whether or not these increases in latency are actually meaningful depends on the application. It is important to note that latencies can by significantly reduced by decreasing the size of the forwarding queues, at the cost of increased losses when burst sizes are large.

Chapter 6

Conclusion

This thesis presents an experimental evaluation of three complementary congestion control strategies for wireless sensor networks. We show that unless a sensor network operating under load has some means of controlling congestion, it will face significant degradation in efficiency and fairness. As network load increases, or when channel variations cause fluctuations in achievable bandwidth, nodes must modulate their send rates based on congestion feedback or the network will go into congestion collapse.

We show that wireless congestion is very different from its wired counterpart. Wired congestion results in buffer drops at internal routers and can be alleviated using such protocols as TCP. Wireless congestion affects networks at a lower level, resulting in drops in the transmission medium. These drops and the resulting degradation in channel quality is caused in part by collisions from hidden terminals as well as by corruption from many distant transmitters. Moreover, congestion in wireless sensor networks substantially reduces fairness and reduces the number of bits delivered per unit energy.

The metrics we use for evaluation express properties that designers and users of sensor networks should find desirable. Network efficiency quantifies the amount of energy that is wasted on transmissions that do not deliver packets. Fairness quantifies the degree of variation in send rates, which impacts sensor network coverage. Imbalance measures the inequity between packet receptions at a given node and its next hop, providing a clearer picture of the hotspots in a network and how that results in buffer and wireless drops. Latency is important because many event-driven applications require the timely reporting of sensor values.

We evaluate three techniques for mitigating congestion both in isolation and in concert. Our results show that hop-by-hop flow control using a simple queue occupancy-based congestion detection method offers substantial efficiency improvements for all types of workloads and utilization levels. This finding holds because a successful wireless transmission requires both the sender and receiver to be contention-free with respect to both the wireless channel and queue space. Implementing a rate limiting policy results in substantial improvements to fairness. Finally, MAC enhancements support the operation of hop-by-hop flow control.

We analyze two ways of detecting congestion: queue occupancy and channel sampling. In addition to offering significantly better performance, queue occupancy requires no support from the MAC layer and is very easy to implement on different platforms.

We present Fusion, a congestion control mechanism that combines hop-by-hop flow control, rate limiting, and a prioritized MAC. Our results show that Fusion improves network efficiency by up to a factor of three and reduces node starvation. Additionally, we show the efficacy of Fusion under a variety of workloads on a 55-node deployment. A simple periodic workload benefits because it is extremely difficult to adequately provision for varied link capacities of a large scale deployment. A high fan-in network realizes gains from congestion control because the nature of that topology makes transit nodes particularly prone to buffer drops. Correlated-event workloads need congestion control to handle the sudden bursts of traffic that spatially-correlated events generate.

6.1 Future work

The results presented in this thesis point to a number of possible areas for future work. First, although the rate limiting scheme presented in Section 4.3 is effective at improving the fairness of networks under load, a robust rate limiting algorithm that correctly handles node failures, skewed routing topologies, and variable send rates would be useful.

Second, even though our implementation of hop-by-hop flow control relies on overhearing, an alternate implementation is possible with the use of link-level acknowledgments to indicate congestion state. Although we have briefly investigated this design point, we leave a performance comparison for future work.

Third, this thesis only analyzed single sensor access point networks. Deploying multiple access points has the potential to decrease network diameter and increase spatial reuse. Although our congestion control techniques only require a few modifications to work in a multiple access point networks, more work would need to be done to adapt supporting routing protocols and path selection metrics to this new architecture.

Finally, while we offer hints as to the sources of loss in our network, more work needs to be done to find definitive answers. In particular, we are investigating whether losses occur because of hidden terminal collisions (in which case RTS/CTS might be a solution) or due to additive interference from distant sources of noise.

Bibliography

[1] Bluetooth SIG. `http://www.bluetooth.org`.

[2] Crossbow Technology Inc. `http://www.crossbow.com/`.

[3] Intel Corporation. `http://www.intel.com/`.

[4] MoteIV Corporation. `http://www.moteiv.com/`.

[5] M. Welsh and G. Werner-Allen. Motelab webpage. `http://motelab.eecs.harvard.edu`.

[6] IEEE 805.15.4 Standard. `http://standards.ieee.org/`, 2003.

[7] I. Aad and C. Castelluccia. Differentiation Mechanisms for IEEE 802.11. In *Proc. of the IEEE INFOCOM Conf.*, pages 209–218, Anchorage, AK, April 2001.

[8] M. Allman, H. Balakrishnan, and S. Floyd. *Enhancing TCP's Loss Recovery Using Limited Transmit.* Internet Engineering Task Force, January 2001. RFC 3042.

[9] Atmel Corporation. ATmega 103L Datasheet. `http://atmel.com/dyn/resources/prod_documents/Doc0945.pdf`.

[10] Atmel Corporation. ATmega 128L Datasheet. `http://atmel.com/dyn/resources/prod_documents/doc2467.pdf`.

[11] H. Balakrishnan. *Challenges to Reliable Data Transport over Heterogeneous Wireless Networks.* PhD thesis, Univ. of California, Berkeley, August 1998.

[12] H. Balakrishnan, H. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. of the ACM SIGCOMM Conf.*, pages 175–187, Cambridge, MA, 1999.

[13] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, Second edition, 1987.

[14] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An Energy-Efficient Coordination Algorithm for Topology Maintainence in Ad Hoc Wireless Networks. In *Proc. of the ACM MOBICOM Conf.*, pages 85–96, Rome, Italy, July 2001.

[15] Chipcon Corporation. CC1000 Transceiver Datasheet. `http://www.chipcon.com/files/CC1000_Data_Sheet_2_2.pdf`.

[16] Chipcon Corporation. CC2420 Transceiver Datasheet. `http://www.chipcon.com/files/CC2420_Data_Sheet_1_2.pdf`.

[17] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proc. of the ACM MOBICOM Conf.*, pages 134–146, San Diego, CA, 2003.

[18] C. Ee and R. Bajcsy. Congestion Control and Fairness for Many-to-One Routing in Sensor Networks. In *Proc. of the ACM SENSYS Conf.*, pages 148–161, Baltimore, MD, November 2004.

[19] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-sync Protocol for Sensor Networks. In *Proc. of the ACM SENSYS Conf.*, pages 138–149, Los Angeles, CA, November 2003.

[20] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proc. of the ACM PLDI Conf.*, pages 1–11, San Diego, CA, June 2003.

[21] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proc. of the ACM MOBICOM Conf.*, pages 56–67, Boston, MA, August 2000.

[22] Intel Corporation. New Computing Frontiers – The Wireless Vineyard. `http://www.intel.com/labs/features/rs01031.htm`.

[23] R. Jain. *The Art of Computer Systems Performance Analysis.* Wiley, First edition, 1991.

[24] D. Johnson, D. Maltz, and J. Broch. *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[25] P. Karn. MACA—A New Channel Access Method for Packet Radio. In *Proc. of the ARRL Conf.*, pages 134–140, London, Ontario, September 1990.

[26] R. Kling, R. Adler, J. Huang, V. Hummel, and L. Nachman. Intel Mote: Using Bluetooth in Sensor Networks. In *Proc. of the ACM SENSYS Conf.*, pages 318–318, Baltimore, MD, 2004.

[27] D. Lee, S. Coleri, X. Dong, and M. Ergen. FLORAX—Flow-Rate Based Hop by Hop Backpressure Control for IEEE 802.3x. In *Proc. of the IEEE HSNMC Conf.*, pages 202–207, Jeju Island, Korea, July 2002.

[28] M. D. Lemmon, Q. Ling, and Y. Sun. Overload Management in Sensor-Actuator Networks used for Spatially-Distributed Control Systems. In *Proc. of the ACM SENSYS Conf.*, pages 162–170, Los Angeles, CA, November 2003.

[29] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler. The Emergence of Networking Abstractions and Techniques in TinyOS. In *Proc. of the USENIX/ACM NSDI Conf.*, San Francisco, CA, 2004.

[30] C. Lu, B. Blum, T. Abdelzaher, J. Stankovic, and T. He. RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks. In *Proc. of the IEEE RTAS Symposium*, pages 55–66, San Jose, CA, September 2002.

[31] H. Luo, S. Lu, and V. Bharghavan. A New Model for Packet Scheduling in Multihop Wireless Networks. In *Proc. of the ACM MOBICOM Conf.*, pages 87–98, Boston, MA, August 2000.

[32] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The Design of an Acquisitional Query Processor For Sensor Networks. In *Proc. of the 2003 ACM SIGMOD Conf.*, pages 491–502, San Diego, CA, 2003.

[33] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proc. of the ACM WSNA Workshop*, Atlanta, GA, September 2002.

[34] P. Mishra and H. Kanakia. A Hop by Hop Rate-based Congestion Control Scheme. In *Proc. of the ACM SIGCOMM Conf.*, pages 112–123, Baltimore, MD, August 1992.

[35] T. Nandagopal, T. Kim, X. Gao, and V. Bhargavan. Acheiving MAC Layer Fairness in Wireless Packet Networks. In *Proc. of the ACM MOBICOM Conf.*, pages 87–98, Boston, MA, August 2000.

[36] W. Noureddine and F. Tobagi. Selective Backpressure in Switched Ethernet LANs. In *Proc. of the IEEE GLOBECOM Conf.*, pages 1256–1263, Rio de Janeiro, Brazil, December 1999.

[37] C. Özveren, R. Simcoe, and G. Varghese. Reliable and Efficient Hop-by-Hop Flow Control. In *Proc. of the ACM SIGCOMM Conf.*, pages 89–100, London, UK, August 1994.

[38] C. Perkins. Ad-hoc on-demand distance vector routing. In *Proc. of the IEEE WMCSA Workshop*, pages 90–100, New Orleans, LA, 1999.

[39] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. of the ACM SIGCOMM Conf.*, pages 234–244, London, UK, August 1994.

[40] L. Perterson and B. Davie. *Computer Networks.* Morgan Kaufmann, 2000.

[41] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *Proc. of the ACM SENSYS Conf.*, pages 95–107, Baltimore, MD, November 2004.

[42] J. Polastre, G. Tolle, and J. Hui. Low Power Mesh Networking with Telos and IEEE 802.15.4. In *Proc of the ACM SENSYS Conf.*, pages 319–319, Baltimore, MD, 2004.

[43] RF Monolithics Corporation. TR1000 Transceiver Datasheet. `http://www.rfm.com/products/data/tr1000.pdf`.

[44] Y. Sankarasubramaniam, O. Akan, and I. Akyildiz. ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *Proc. of the ACM MOBIHOC Conf.*, pages 177–189, Annapolis, MD, June 2003.

[45] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *Proc. of the ACM SENSYS Conf.*, pages 1–12, Baltimore, MD, 2004.

[46] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *Proc. of the IEEE SNPA Workshop*, pages 102–112, Anchorage, AK, May 2003.

[47] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *Proc. of the ACM SENSYS Conf.*, pages 214–226, Baltimore, MD, 2004.

[48] Texas Instruments. MSP430 Datasheet. `http://www-s.ti.com/sc/ds/msp430fw427.pdf`.

[49] University of California, Berkeley. Firebug. `http://firebug.sourceforge.net/`.

[50] T. van Dam and K. Langendoen. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proc. of the ACM SENSYS Conf.*, pages 171–180, Los Angeles, CA, November 2003.

[51] C. Wan, A. Campbell, and L. Krishnamurthy. PSFQ: a Reliable Transport Protocol for Wireless Sensor Networks. In *Proc. of the ACM WSNA Workshop*, pages 1–11, Atlanta, GA, 2002.

[52] C. Wan, S. Eisenman, and A. Campbell. CODA: Congestion Detection and Avoidance in Sensor Networks. In *Proc. of the ACM SENSYS Conf.*, pages 266–279, Los Angeles, CA, November 2003.

[53] D. G. Willard. A Time Division Multiple Access System for Digital Communication. In *Computer Design*, pages 79–83, June 1974.

[54] A. Woo and D. Culler. A Transmission Control Scheme for Media Access in Sensor Networks. In *Proc. of the ACM MOBICOM Conf.*, pages 221–235, Rome, Italy, 2001.

[55] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proc. of the ACM SENSYS Conf.*, pages 14–27, Los Angeles, CA, November 2003.

[56] Y. Xu, J. Heidemann, and D. Estrin. Geography-Informed Energy Conservation for Ad Hoc Routing. In *Proceedings of the ACM MOBICOM Conf.*, pages 70–84, Rome, Italy, July 2001.

[57] F. Ye, H. Luo, S. Lu, and L. Zhang. Poster Abstract: Statistical En-route Filtering in Large Scale Sensor Networks. In *Proc. Of the ACM SENSYS Conf.*, pages 330–331, Los Angeles, CA, November 2003.

[58] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proc. of the IEEE INFOCOM Conf.*, pages 1567–1576, New York, NY, June 2002.

[59] Y. Yi and S. Shakkottai. Hop-by-hop Congestion Control over a Wireless Multi-hop Network. In *Proc. of the IEEE INFOCOM Conf.*, Hong Kong, June 2004.

[60] J. Zhao and R. Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks. In *Proc. of the ACM SENSYS Conf.*, pages 1–13, Los Angeles, CA, November 2003.