Wireless Sensor Network Metrics for Real-Time Systems

Phoebus Wei-Chih Chen

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-75
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-75.html

May 20, 2009

Acknowledgement

This dissertation is supported by ARO's MURI program "Urban Target Recognition by Ad-Hoc Networks of Imaging Sensors and Low-cost, Nonimaging Sensors," under contract number W911NF-06-1-0076, and by TRUST (Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: AFOSR (#FA9550-06-1-0244), BT, Cisco, ESCHER, HP, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, Telecom Italia, and United Technologies.

Wireless Sensor Network Metrics for Real-Time Systems

by

Phoebus Wei-Chih Chen

B.S. (University of California, Berkeley) 2002
M.S. (University of California, Berkeley) 2005

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor S. Shankar Sastry, Chair
Professor Kristofer Pister
Professor Raja Sengupta

Spring 2009

The dissertation of Phoebus Wei-Chih Chen is approved.

Chair	Date
	Date
	Date

University of California, Berkeley

Wireless Sensor Network Metrics for Real-Time Systems

by

Phoebus Wei-Chih Chen

Abstract

Wireless Sensor Network Metrics for Real-Time Systems

by

Phoebus Wei-Chih Chen

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor S. Shankar Sastry, Chair

Research in wireless sensor networks (WSNs) is moving from studies of WSNs in isolation toward studies where the WSN is treated as a component of a larger system. One hot area of research is the study of how to integrate wireless sensor networks with existing real-time measurement and control systems, forming what we call a *wireless networked system*. To apply the theories for studying real-time systems operating over wireless networks, for instance the theory of Networked Control Systems, we need to develop models of the lossy wireless communication medium. The models are used to compute *network metrics*, measures of network performance such as latency and reliability, that serve as an abstraction of the network which are used as input to the theoretical tools for analyzing the entire wireless networked system.

This dissertation focuses on modeling WSNs which use mesh networking with a TDMA data link layer. Specifically, it focuses on two classes of TDMA mesh networking schemes, *Unicast Path Diversity* (UPD) and *Directed Staged Flooding* (DSF). UPD uses retransmissions to get reliable packet delivery while DSF uses constrained flooding / multicast to get reliable packet delivery. We derive Markov chain models of UPD and DSF to compute the probability of end-to-end packet delivery as a function of latency, the expected radio energy consumption on the nodes from relaying packets, and the traffic distribu-

tion on the network. We also derive metrics based on clearly defined link failure models and routing models that allow a network designer to compare mesh routing topologies to determine which is better for reliable packet delivery. One of these network metrics leads to a greedy algorithm for constructing a mesh routing topology. Finally, we study the implications of using distributed scheduling schemes to generate schedules for WSNs. Particularly, we focus on the impact scheduling has on path diversity, using short repeating schedules and Greedy Maximal Matching scheduling as examples. A cluster-based scheduling scheme is proposed which works well on a subclass of network topologies.

Professor S. Shankar Sastry
Dissertation Committee Chair

To my Mom and Dad, who have supported me throughout the years.

Contents

List of Figures

List of Tables

Acknowledgements

First off, thanks to my advisor Professor Shankar Sastry for supporting my research and giving me the freedom to explore my interests. In addition to technical knowledge, I've learned a lot about myself over these years in graduate school — how to motivate myself when I get stuck, under what working conditions am I most productive, what types of problems excite me, what do I value in life, etc. I would also like to thank my colleague Songhwai Oh who has served as a research mentor throughout my graduate career. He really helped me "learn the ropes" about the research publication process and how to formulate research problems and define the scope of one's research.

In my earlier years of graduate school, I got a lot of advice about graduate school and research in general from the postdocs in my cubicle, Jonathan Sprinkle and Rick Groff (and a big thanks to Jon for lending me his LaTeX manual in times of need!). I also got advice from my fellow grad student Parvez Ahammad, who was always open to sharing his stories and discoveries about the process of doing research and surviving in the academic world. These stories really provided some moral support when times got tough and research was not going well.

Professor Kris Pister provided many valuable insights on the state of affairs of wireless sensor networks in industry. The problems they were facing with multi-path fading, and the experiments demonstrating that frequency hopping on the 802.15.4 radio was an effective countermeasure, helped shape the direction of my dissertation research. His company's protocol, TSMP, is the basis for many of the models and analysis tools in this dissertation.

Professor Jean Walrand listened to me explain my research in its earlier stages and provided references to other related works. Professor Kannan Ramchandran saved the day by sitting in on my qualifying exam committee despite being given last minute notification that one of my committee members had an emergency and could not attend. He also suggested that I include energy evaluation into my models, which now appears in this dissertation.

My EE224B partner, Ian Tan, was the first person to vet my ideas for modeling Directed

Staged Flooding (although it wasn't called that back then!). Sinem Ergen also listened to me talk about my research during its middle stages and provided helpful comments and additional references.

I got a lot of support from many of the staff members in Professor Sastry's group: Sally Alcala, Jessica Gamble, Gary Givens, Maria Jauregui, Gladys Khoury, and Mary Stewart. They kept and eye out for me and went out of their way to help me navigate through all the different processes of the University. I also got a lot of support from several of the Computer User Support Group veterans: Ann DiFruscia, Phil Loarie, and Lars Rohrbach. I often had computer related requests and issues, and these guys were quick to come to the rescue. Ruth Gjerde, the graduate student affairs advisor, provided a calm and warm environment to all of the graduate students — she was like a Mom for all of us. I especially remember and appreciate her help when I was frantically looking for an advisor at the end of my first year of graduate school.

I really enjoyed interacting with the other members of the Sastry group. They provided good conversation and a pleasant research environment. There are too many to name here, but I will mention three in particular because their names sound really cool in sequence: Edgar, Allen, Po. If only "Po" would be spelled "Poe." Of course, there were also many other graduate students outside my research group with whom I've had helpful discussions regarding career directions and the academic research environment.

Outside research, I've had good support from friends, family, and mentors that helped me make it through graduate school. I have to thank my Mom and Dad, and my brother Alex. I also had good roommates and friends from my Wushu and Taichi classes. Thanks to Sifu Bryant Fong and Sifu William Dere for instructing the classes. They were a good way to relax and take a break from work. I also had fond memories of cooking and making "dumplings the size of my fist" with Mei Mei and company.

Last, thanks to Anande Sawarte and Matthew A. d'Alessio for the LaTeX template.

Parts of this dissertation were taken and updated from a book chapter written by myself,

Songhwai Oh, and Shankar Sastry in the book "Homeland Security Technology Challenges: From Sensing and Encrypting to Mining and Modeling" published by Artech House.

This dissertation is supported by ARO's MURI program "Urban Target Recognition by Ad-Hoc Networks of Imaging Sensors and Low-cost, Nonimaging Sensors," under contract number W911NF-06-1-0076, and by TRUST (Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: AFOSR (#FA9550-06-1-0244), BT, Cisco, ESCHER, HP, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, Telecom Italia, and United Technologies.

Curriculum Vitæ

Phoebus Wei-Chih Chen

Education

2005	University of California Berkeley
	M.S., Electrical Engineering and Computer Sciences
2002	University of California Berkeley
	B.S., Electrical Engineering and Computer Sciences
1998	Manalapan High School
	Science and Engineering Magnet Program

Personal

Born	1980, USA

Chapter 1

Introduction

Wireless Sensor Networks (WSNs, or sometimes just *sensor networks*) consist of low-power embedded devices with sensing, computing, a power source, and a wireless radio. These devices, called *sensor nodes* or *motes*, are often battery powered and placed throughout an environment to sense and relay processed measurements over a wireless network to a server. WSNs enable a large variety of applications to gather data from multiple sensors without the high cost of wiring. For instance, WSN applications for monitoring and surveillance include habitat monitoring [104; 108; 39], monitoring the health of industrial equipment for maintenance [53], monitoring the stress on large civil structures such as bridges and buildings [51; 114], and supply-chain tracking of shipping containers [67].

Although these applications form the bulk of WSN applications today, WSNs are not limited to monitoring and surveillance. In fact, the focus of this dissertation is on how WSNs can be integrated with other systems that respond to sensed changes in the environment, forming what we call *wireless networked systems*. A wireless networked system thus consists of the wireless sensor network, a decision-making system (e.g., an automatic controller or a human operator), and the environment that we are sensing and responding to (e.g., a manufacturing plant, highway traffic, forests under wildfire surveillance). Note that the distinction between these components in a wireless networked system are conceptual. In reality, the physical boundaries between the components can be blurred: the decision-

making entity may be the motes themselves, and the motes may be considered a part of the environment in scenarios where the motes are physically connected to mobile robots.

1.1 Challenges in Designing Wireless Networked Systems

The integration of wireless sensor networks into wireless networked systems has been challenging because WSNs have only become mainstream in the last decade and researchers are still grasping how to use them effectively and efficiently. Research on WSNs covers a wide array of topics such as energy-efficient communications, network self-organization and leader election, data aggregation, sensing coverage and placement, security and privacy, and others [22; 32]. The advances in some of these topics have directly improved the performance of real-world WSN deployments, but major challenges remain which have slowed the commercial adoption of WSNs. Among the biggest challenges are reliable communications and modeling / characterizing the sensor network so it can tightly interface with other components in the encompassing wireless networked system.

WSNs must communicate over a time-varying wireless channel that can only be characterized by probabilistic models because the environmental conditions affecting the wireless channel are difficult to measure exactly. This is a general problem in wireless communications and as a result we can only get *probabilistic guarantees* on network performance. In fact, many WSNs built today (and many wireless communication systems) are "best effort" systems, meaning they provide little insight and few guarantees to the designers on the throughput, latency, or reliability of the communication. As a result, systems have to be built first, then tested and tweaked later. This ad-hoc design methodology does not work when we scale up the number of nodes in the WSN or when we integrate the WSN into a complex system.

A more structured design methodology starts by examining the system requirements of the wireless networked system and then extrapolating these requirements to obtain design guidelines for the WSN. Where possible, the WSN should run communication algorithms that are amenable to modeling, particularly models that provide *metrics* on the perfor-

mance of the WSN to the rest of the wireless networked system. If we treat the wireless networked system from a *Control Systems* perspective, then such models let the decision-making entity account for network conditions when regulating the system's behavior, and allows the designer to predict the entire system's performance.

One system requirement which will guide the study of wireless networked systems in this dissertation is the communication timing requirement between the components of the system. Systems such as traditional wireless sensor networks set up for environmental monitoring and offline data analysis [104; 108] have loose latency requirements, on the order of hours to send data to a collection point. Systems such as fire alarm systems in buildings have tighter latency constraints, less than 90 seconds to report a fire [57] and preferably an even faster response time to turn off dampers in the ventilation system and stop the spread of smoke to other parts of the building [100]. Typically, tight latency constraints are required in systems that are involved in *real-time decision making*, whether the decisions are in response to an event / alarm or to help continuously regulate a system. This dissertation will focus on wireless networked systems with tight timing constraints, particularly wireless *real-time measurement systems* and wireless *real-time control systems*.[1] These will be discussed in detail below in Sections 1.1.1 and 1.1.2.

1.1.1 Real-Time Measurement Systems

A real-time measurement system consists of sensors and processors that perform signal processing, estimation, or inference on the readings from the sensors before delivering it to the user. The components in the system may be connected by a network, but the communication delay must be low (Figure 1.1). The user is often a human who needs to take an action or make a decision given the data from the system.

Wireless sensor networks can be used as real-time measurement systems if they are designed to ensure data is delivered with low latency. Most current wireless sensor network deployments do minimal processing to route and aggregate data, and as such can be modeled

[1] We use the term *real-time* to mean that low latency is important to our systems. Because we are dealing with statistical models of networks, we do not provide *hard guarantees* of meeting real-time constraints, as is often the goal of research in the area of *Real-Time Systems*.

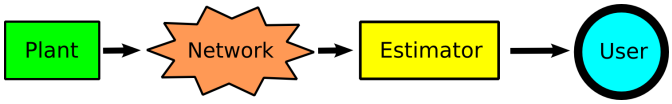

Figure 1.1. A real-time wireless networked measurement system.

as simply a set of sensors connected by a network. This is very similar to the simple model of real-time control systems covered below in Section 1.1.2. As the sensor node hardware improves and the algorithms become more complex, we will also need to model the in-network processing of data by the sensor network.

Fast data collection and reliable data delivery are some of the most critical and challenging design objectives for building real-time measurement systems with current sensor networks. These two objectives depend on the networking protocols used in the sensor network. Furthermore, to characterize application performance, we need good models of these networking protocols for predicting reliability and latency. This will be the subject in Chapter 4.

1.1.2 Real-Time Control Systems

From a control systems point of view, a wireless networked system consists of a plant / system to be controlled, sensors, actuators, and computers connected together by a network that is wireless or partially wireless (See Figure 1.2). The distinction of real-time control systems from real-time measurement systems is that computers, instead of humans, make decisions to regulate the plant. Latency and reliability are still critical network performance parameters, but at the same time a theory needs to be developed to address how to design automatic controllers in these systems. This is the theory of *Networked Control Systems* presented later in Section 4.1.

The key issue in designing a wireless networked control system is addressing how to guarantee system performance and stability when the communication medium is lossy and delivers data with variable delay. In some systems such as distributed heating and ventilation of office buildings, it may be sufficient to guarantee system performance in a statistical sense — for instance, the system works 99.9% of the time. However, it would be foolish to

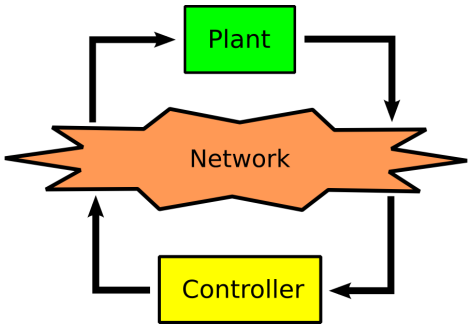

Figure 1.2. A real-time wireless networked control system.

build safety-critical systems over a wireless network without a contingency plan when the wireless channel between the system components is disconnected. Putting in a redundant, wired communication channel may be impossible or too costly. However, we may be able to design a safety-critical control system that continues to perform safely, but with lower performance, without a connected network and performs *better* with a wireless connection. Equivalently, in automated manufacturing, the system can be designed to maintain product *quality* but vary the *yield* depending on the reliability and delay of the wireless network.

Figure 1.3 proposes an abstract scenario and a handshaking protocol for the components of a wireless networked system to coordinate and maintain safety / product quality while adapting to wireless network conditions to maximize performance. In this scenario, the output of system A is being fed as input to system B, with some delay D. While system B can measure its immediate inputs and perform local feedback control, it can get better performance with a larger look-ahead horizon by getting direct measurements of the output of system A. System A can measure its outputs and send it over the wireless network to system B, and furthermore it can adjust the rate of its output in response to how well it can coordinate with system B. If it does not hear an end-to-end acknowledgment from system B, system A will assume that system B did not receive the measurements at the output of system A, and slow down it's output accordingly. This way, the stability of the system or the quality of the output will not be compromised, but the performance or yield can be tuned based on wireless network conditions.

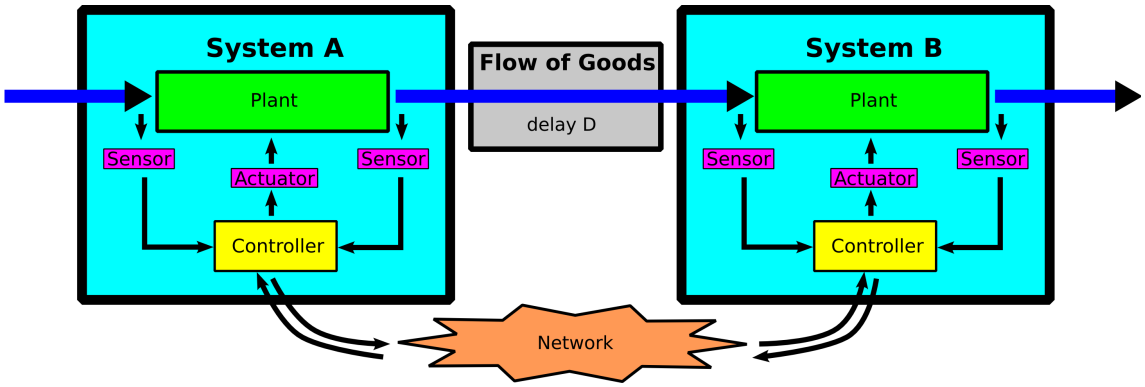

Figure 1.3. A sample scenario and handshaking protocol for coordinating the components of a wireless, networked, safety / quality-critical control system. See text for details.

1.2 Problem Statement and Contributions

Currently, there is a need for a comprehensive metrics-based methodology to design and validate wireless networked systems. Existing approaches often design the real-time measurement or real-time control system in isolation from the design of the underlying wireless network. The theory of Networked Control Systems (NCSs) attempts to bridge these two design spaces by abstracting the network as a simple point-to-point channel and studying how packet loss, variable delay, variable sampling, and quantization error affect the closed-loop control system. In order to apply this theory to wireless networked systems, we need to develop network models where we can extract the relevant parameters for this abstraction. We call these parameters the *network metrics*, and they serve as an *interface* for tuning the NCS to network conditions and tuning the network to meet the requirements of the NCS.

Traditional network metrics such as throughput and fairness are tailored for efficient delivery of files between users and not tailored for real-time decision making systems. The main focus of this dissertation is to develop new network metrics and models of WSNs to use in the design of wireless networked systems. This involves both developing algorithms to compute the metrics and using the metrics to help identify how to improve the system. For instance, these metrics may be used to help construct better routing topologies and schedules. The metrics in this dissertation focus on reliability (i.e., the likelihood that a packet is delivered to the destination) and latency.

The contributions of this dissertation are as follows:

- Chapter 2 develops a set of metrics to measure the end-to-end reliability of a routing topology. It also presents a metrics-based routing topology generation scheme.
- Chapter 3 studies the effects of various scheduling schemes on packet latency and packet *path diversity*. It also presents a distributed scheduling scheme designed for *layer-to-layer* routing topologies.
- Chapter 4 develops an end-to-end connectivity metric given a routing topology and TDMA schedule for the wireless network. It also develops Markov chain models for two classes of TDMA wireless networking protocols: *Unicast Path Diversity* and *Directed Staged Flooding*. These Markov chain models are also used to compute the traffic distribution throughout the network and estimate the radio energy consumption on the nodes from relaying packets. The chapter concludes with two in-depth case studies applying these tools on a simulated real-time measurement system and a simulated real-time control system.

The new terminology used above will be defined in the respective chapters.

This dissertation models the WSN as a TDMA wireless mesh network. As will be explained in Section 1.3.3, we focus on TDMA networks for ease of modeling and mesh networks for better path diversity and reliability. The WSN does not do in-network data aggregation or processing, so it simply behaves as a self-organizing communication network (self-organizing in the sense that the WSN constructs its own routing topology and schedule). We focus on the scenario where there are few decision makers (e.g., controllers or human operators) in the network. This means there are very few sessions or flows in the network.

1.3 Background and Related Work on Wireless Networked Systems

This section presents some examples of wireless networked systems and background on wireless networking standards and WSN radio hardware specifications to relate the content of the following chapters to concrete applications and current technologies. Since the chapters span such a wide range of topics, the related work for the relevant topics will be reviewed at the beginning of each chapter.

1.3.1 Examples of Real-Time Measurement Systems

There are a large variety of applications for real-time measurement systems. For instance, operators of video surveillance cameras in office buildings and security alarm systems for banks need to react quickly to intruders, on the order of minutes to dispatch security personnel or lock down critical areas. There have also been proposals to use sensor networks to monitor the vital conditions of the injured in large disaster-response scenarios [99] and to monitor the sick and elderly for assisted living at home [1]. In scenarios where a patient is losing blood and his heart rate drops or an old man falls down the stairs, we would also like to respond and dispatch emergency medics on the order of minutes.

There have also been proposals to use sensor networks as real-time measurement systems to help firefighters navigate through a burning building [112; 101]. The sensor network would keep track of the location of the firefighters and use smoke and temperature sensors to identify the location of the fires and safe evacuation routes, relaying all this information back to the incident commander to coordinate the firefighting efforts. Here, the system response time should be on the order of seconds to match the speed at which the fire can spread and the firefighters can move.

Another real-time measurement system is the *Supervisory Control and Data Acquisition* (SCADA) system used in industrial control systems [36]. SCADA system architectures are typically hierarchical, with localized feedback control and wide area monitoring for diag-

nostics and safety [74], often with a "human in the loop." The diagnostics network needs to relay large amounts of data, and is usually not as sensitive to delay as the safety and control networks. However, diagnostics information can also be used to "close the loop" for equipment shutdown or continuous process improvement, though the actuation is typically event-driven as opposed to time-driven continuous actuation. The safety networks require determinism (guaranteed response time), low delay, and reliable data delivery, though the traffic load may be lower than diagnostics networks. The response time / delay requirements depend on the particular industrial plant, and can range from microseconds to hours, depending on the task [69].

Finally, real-time measurement systems can be used for battlefield awareness in military applications. For instance, there is an interest in using sensor networks to track moving targets [76]. The tolerable delay in such a system depends on the distance of the target to critical areas, on the order of seconds to tens of seconds. Sensor networks can also be used to locate snipers from the acoustic trail of bullets and the muzzle blast of guns [56]. Such systems may need time synchronization between sensor nodes on the order of microseconds to compute sniper locations, but they can tolerate communication delays to the user on the order of seconds.

1.3.2 Examples of Real-Time Control Systems

Real-time control systems are often part of SCADA systems for process control and industrial automation. Traditional applications range from chemical production and petroleum refinery to waste management. Current research is trying to extend these systems to reconfigurable manufacturing systems that can change production capacities to quickly match consumer demands [72; 74]. In these systems, wireless connections between equipment can reduce the cost of reconfiguration.

A good application of wireless sensor networks to real-time control systems is the control of lighting and Heating, Ventilation, and Air Conditioning (HVAC) systems. Studies have shown that 38% of all primary energy use is devoted to conditioning residential and

commercial buildings, making it the largest sector of energy consumption, even exceeding transportation and industry [111]. Furthermore, the Department of Energy estimates that half of the total energy use can be avoided, resulting in considerable savings. Here, the latency requirements are on the order of seconds and minutes, and slower system response times only results in less efficiency, not a compromise of safety.

Another real-time control application for wireless sensor networks is active / semi-active damping of civil structures [33; 95]. Active damping is the actuation of parts of a structure to dampen vibrations, while semi-active damping changes the dissipation properties of passive dampers in real-time. This can be useful for reducing the damage to large buildings and bridges during earthquakes or hurricanes. The latency requirements for such systems are on the order of hundredths of a second. Wireless sensor networks enable existing structures to be retrofitted with active / semi-active dampers without the installation of wires, while simultaneously monitoring them for structural damage [51; 114].

Wireless sensor networks are also well suited to real-time control applications with mobile components, particularly robotics. Some experiments have shown that wireless sensor networks can be used to help unmanned ground vehicles navigate through unknown terrain and follow targets [61; 98]. Here, the timing requirements are on the order of tenths of a second for obstacle avoidance and seconds for path planning and target following.

1.3.3 Wireless Networking

The type of wireless networks connecting our real-time control and measurement systems are *packet radio networks* (PRNs) [6]. In a packet radio network, data is digitized and broken up into units called *packets* before being transmitted over a multiaccess channel. Different subsets of nodes can communicate with each other on this channel and packet transmissions may collide, causing the packets to be dropped. Most wireless sensor networks today are packet radio networks.

To compute our wireless network metrics, we will model the packet radio network at the *network* and *data link* (sometimes known as media access, or MAC) layers of the seven layer

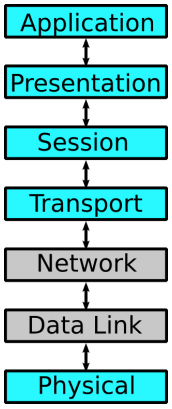

Figure 1.4. The seven layer *Open Systems Interconnect* (OSI) model of networks. For more details, see a textbook on communication networks, such as [6].

OSI model (See Figure 1.4). We select implementations of these OSI layers that enable the wireless network to provide high reliability while being easy to model.

Modeling Assumptions

We make several simplifying modeling assumptions to tailor the wireless network to real-time control and measurement systems. Some of these assumptions will be mentioned again in the context of the models they affect later in the dissertation.

While we do not explicitly model the physical layer of the network, the wireless nature of the links affects our models. First, we assume links are mutually independent and independent over time. This is a rather strong assumption, but it is necessary for making our models tractable. It is partially justified by frequency-hopping and distributing link transmissions over time, as explained in Chapter 4. We do not assume that all the wireless links are *bidirectional* (*symmetrical*), meaning we can have a situation where node i can transmit to node j but node j cannot transmit to node i. Similarly, we can have asymmetric interference between nodes, where node i's transmission interferes with node j's reception, but not vice versa. However, our networking algorithms will only select bidirectional links for transmission so the receiver can send acknowledgment messages to confirm that a packet was successfully transmitted. We do not use the locations of the nodes to derive the connec-

tivity and interference graphs for the network.[2] Instead, we assume that the connectivity and interference graphs are provided to us by link estimators running on the network.

We are primarily concerned with continuous data traffic as opposed to bursty data traffic for real-time control systems. Of course, alarms and event-generated traffic need to be handled, but we assume these are infrequent and can be provisioned ahead of time. As a result, we will focus on *proactive* (as opposed to *reactive* or *on-demand*) routing protocols, meaning that routes are established and maintained before they are used to send data.

Also, some real-time measurement systems may have multiple traffic flows (also called *sessions*) with different bandwidth requirements. For instance, wireless networked systems with camera sensors [17] may have high-bandwidth, single-hop traffic between the nodes to transfer image features for computing image correspondence on cameras sharing the same field of view. In addition, there may be low-bandwidth, multi-hop traffic to the user to report short, high-level descriptions of what is being observed in the environment. The network must support multiple simultaneous sessions.

Since we are focused on wireless networked systems with few decision makers and many wireless sensor nodes, we will focus on many-to-one and one-to-many routing for collecting measurements and disseminating commands. As we shall see in Chapter 2, we will exploit multiple paths between nodes in the network to provide better end-to-end reliability. We also assume that the network topology is relatively static and stable, meaning we do not explicitly model node mobility or the association and disassociation (also known as joining and leaving) of nodes. The network, because of multi-path routing, should be robust to the failure of a small number of nodes.

[2]This means we do not assume a disk communication model (all nodes within a radius of node i can communicate with node i) and do not assume that signal strength decays over distance depending on a path loss exponent. These assumptions may be useful in other works for analyzing the computational complexity of a networking algorithm.

Physical Layer

The models in this dissertation are mostly agnostic to the details of the physical layer used in the wireless network. However, the range and magnitudes of the data rates and packet sizes give a sense of how our models will apply in practice.

Many wireless sensor network radios today conform to the IEEE 802.15.4 standard [55], which is a PHY and MAC layer specification for personal area networks (PANs). The standard is targeted at low-power, low-bandwidth, short range ($\approx 10 - 100$ m) radios. The specification targets three bands of open spectrum operating frequencies:

- 868.3 MHz, with 1 frequency channel at 20 kbps (kilobits per second)
- 902-928 MHz, with 10 frequency channels at 40 kbps
- 2.4-2.4835 GHz, with 16 frequency channels at 250 kbps.

The radio can scan the frequency channels to select a channel with low outside interference, or *frequency hop* over multiple frequency channels (known as *frequency hopping spread spectrum* in the literature). Many WSN radios operate in the 2.4 GHz band because of the higher data rate (which translates to less power per bit transmitted) and the availability of more channels. The reported bit rates (e.g., 250 kbps) are the raw bit rates of the radio, not the payload bit rates.

Table 1.1 summarizes the theoretical data rates and packet sizes when using different combinations of PHY and MAC headers. The PHY header is 6 Bytes and the MAC header can range from 9 to 25 Bytes. Of course, in reality transmissions need to allocate

- clear channel assessment (CCA) time[3] (discussed below),
- turnaround time[4] to switch the radio from receiving to transmitting and vice versa,
- (optional) guard times to accommodate for clock drift between the transmitter and receiver,[5] and

[3] 802.15.4 specifies a minimum CCA time of 128 μs at 2.4 GHz.
[4] 802.15.4 specifies a maximum turnaround time of 192 μs at 2.4 GHz.
[5] This is necessary if the transmitter and receiver need to coordinate to rendezvous at a TDMA time slot.

Table 1.1. Theoretical data rates and packet sizes for 802.15.4 radio at 2.4 GHz

Packet Type	Packet Size (Bytes)	Payload Size (Bytes)	1 Packet Tx Time (ms)	Packet Rate (pkts/sec)	Payload Rate (Bytes/sec)
empty payload	6	0	0.192	5208.3	0
min size ACK	11	0	0.352	2840.9	0
max size (no MAC header)	133	127	4.256	234.96	29840
max size, 2 Byte addr in MAC	133	127 - 9 = 118	4.256	234.96	27725
max size, 8 Byte addr + 2 Byte PAN ID in MAC	133	127 - 25 = 102	4.256	234.96	23966

- (optional) additional time to calculate *cyclic redundancy checks* (CRCs) and *message integrity codes* (MICs).

The 802.15.4 physical layer specification includes useful services for assessing interference and signal quality on the channel, which can be used for link estimation and assigning link costs by upper layers. These include the *Receiver Energy Detection* (ED), or *Received Signal Strength Indicator* (RSSI), which estimates the total signal strength in the bandwidth of the channel and the *Link Quality Indicator* (LQI) which estimates the quality of a received packet. Also, the radio performs a *clear channel assessment* before sending a packet, which can also be used by upper layers to assess the amount of outside interference and contention / congestion in the network.

Data Link Layer

At the data link / MAC layer, we will model *time division multiple access* (TDMA) media access schemes, where nodes are scheduled to transmit during time slots. The other popular media access scheme used by the simple radios in sensor networks, *carrier sense multiple access* (CSMA), is not studied here because of the difficulty of accurately modeling random collisions when multiple nodes simultaneously try to access the medium. The

variants of TDMA schemes (e.g., LEACH [41], BMA [60], LMAC [110], TRAMA [82], TSMP [80]), essentially differ in the protocols for scheduling the nodes. There are also variants that are hybrid CSMA / TDMA schemes to improve performance while avoiding some of the complexity of optimal scheduling in TDMA, such as Scheduled Channel Polling (SCP) [117] and Z-MAC [85], but for modeling simplicity we stick to the most basic TDMA model.

Some commercially available sensor networks follow the 802.15.4 MAC specifications. 802.15.4 specifies both a CSMA and a TDMA mode of operation as well as mechanisms for nodes to associate and disassociate with the network through a *PAN coordinator* (a node responsible for coordinating the network). Some of the MAC layer mechanisms may be implemented in software and not by the radio chip hardware. For instance, on sensor network platforms which use the CC2420 radio chip [19] (a popular chip used by many platforms), the implementation of the backoff and retransmission mechanisms in CSMA and the allocation of *Guaranteed Time Slots* (GTS) in TDMA is the responsibility of the microcontroller controlling the CC2420 radio chip. In fact, some sensor networks do not implement the full functionality of the 802.15.4 MAC layer. For instance, open source sensor network operating systems such as TinyOS [109] allow researchers to tinker with the mechanisms in the MAC protocol while conforming to the general 802.15.4 packet format.

The models in this dissertation can be applied to TDMA protocols implemented on 802.15.4 radios, including ones that do not conform to the details of the 802.15.4 specifications to allocate guaranteed time slots. At the MAC layer, we only model that

- the network is time synchronized,
- nodes send and receive packets during time slots,
- a node can only receive from one other node in a time slot (one radio per node), and
- there can be acknowledgments and retransmissions on a link.

Specifically, the models do *not* cover these mechanisms of the 802.15.4 MAC standard:

- generation of network beacons by a PAN coordinator

- synchronization to network beacons
- PAN association and disassociation
- encryption and security
- CSMA-CA mechanism for channel access

Wireless Networking Standards for Upper Layers

There are many efforts to define wireless networking standards operating in the layers above the IEEE 802.15.4 PHY / MAC layer. A large industry consortium was formed to develop the Zigbee network and application layer standard [119]. Zigbee is initially targeted at energy management and efficiency, home automation, building automation, and industrial automation.

A Zigbee network is initially created by a node known as the *network coordinator* and each network operates over a *single frequency channel* (no frequency hopping). A node can join the network as a *router* or an *end device* which does not participate in routing packets for other nodes. A node joins the network as a child of an existing node, thus forming an *association tree* with the network coordinator at the root of the tree. Nodes can be assigned addresses stochastically, meaning they pick an arbitrary address and then resolve any addressing conflicts later, or they can be assigned addresses hierarchically, meaning they are assigned addresses which reflect where they are in the tree (all descendants of a node reside in a predefined block of addresses). The latter is used to help routing.

Zigbee supports many-to-one routing to a *concentrator*, one-to-many routing either to a group (also known as *multicast*) or the entire network (broadcast), or one-to-one routing (*unicast*) between nodes. In one-to-one routing, packets travel down a single path while in multicast packets travel down a single path until they reach a member of a group, at which point a packet switches to broadcast to reach the remaining members of the group. Routes can be built on-demand and stored in the routing tables of the nodes on the routing path. Route discovery is not necessary if the network uses hierarchical addressing since

the network can use *hierarchical / tree routing* to route packets along the edges of the association tree. Zigbee also supports *source routing*, where the entire path of a packet is stored in the packet.

Zigbee also specifies that nodes need to broadcast link status update messages to their neighbors periodically to ensure good links are used for routing. In fact, a node can only join the network if the link cost to its intended parent in the association tree is below a threshold (the default threshold is 3). Zigbee specifies that the (integer) link cost $c_l \in [1, 7]$ can be fixed ($c_l = 7$) or can be computed via the formula:

$$c_l = \min(7, \text{round}(\frac{1}{p_l^4})) \tag{1.1}$$

where p_l is the probability of reception on the link. The decision of which link cost to use is up to the implementer, and the method of measuring p_l (e.g., using LQI or estimating the packet reception rate by tracking sequence numbers of packets) is also up to the implementer. If the implementer wishes to enforce that links are symmetric, then the cost of the link $l = (i, j)$ (and $l = (j, i)$) is $\max(c_{(i,j)}, c_{(j,i)})$. The cost of a path is simply the sum of the costs of the links on the path. The route discovery algorithm selects paths with the minimum cost.

Zigbee's use of simple link costs and a single path to route between nodes (except for broadcasts) raises the concern that routing between nodes may be unreliable. On the other hand, the WirelessHART [40] standard for process measurement and industrial automation is based off a multi-path routing protocol. WirelessHART is also a network and application layer standard built on top of the IEEE 802.15.4 PHY / MAC standard. WirelessHART uses TDMA, with 100 time slots per second, and frequency hops for each packet transmission. It supports broadcast, multicast, and unicast transmissions as well as source routing. WirelessHART also supports different traffic priority levels, reservation of bandwidth for traffic, simple end-to-end acknowledgment of data delivery, and a TCP-like reliable transport handshaking protocol. WirelessHART is based off of Dust Network's TSMP protocol, which we will discuss in more detail in Chapter 4.

ISA100.11a [45] is yet another wireless standard for industrial automation. Release 1 of

the standard is built on the PHY layer provided by IEEE 802.15.4, and is currently in the approval process as of the writing of this dissertation. The standard is meant to incorporate existing application protocols (originally for wired buses) for process monitoring and control such as native ISA100, HART, Foundation Fieldbus, DeviceNet, Profibus, Modbus, CIP, and others. It also features TDMA mesh routing with frequency hopping for greater reliability. The significant efforts of ISA (The International Society of Automation) to develop this standard shows that the industrial automation community expects wireless real-time control and measurement systems to play an important role in the future.

1.4 Basic Graph Theory Background and Notation

We will be using some basic concepts from Graph Theory throughout this dissertation. A summary of the math notation used in the chapters can be found in Appendix A. For a more thorough treatment of Graph Theory, see [37].

A *graph* $G = (\mathcal{V}, \mathcal{E})$ consists of a set of *vertices* (*nodes*) $\mathcal{V} = \{1, \ldots, N\}$ and a set of *edges* (*links*) $\mathcal{E} \subseteq \{(i,j) : i,j \in \mathcal{V}\}$. In this dissertation, unless stated otherwise, the order of the nodes in the ordered pair representing an edge is important, i.e., $(i,j) \neq (j,i)$. Thus, unless stated otherwise, we use G to represent a *directed graph* with *directed edges*. In an *undirected graph*, $(i,j) = (j,i)$. An undirected graph can be represented by a directed graph where if $(i,j) \in \mathcal{E}$ then also $(j,i) \in \mathcal{E}$.

We say two nodes i and j are *adjacent* if there exists an edge (i,j) or (j,i) in $\mathcal{E}$. We say that a node is *incident* on an edge if it is one of the edge's two vertices. The *neighbors* of a node i are represented by the set of nodes $\mathcal{N}_i = \{j : \exists\,(i,j) \text{ or } (j,i) \in \mathcal{E}\}$. The *incoming links* of a node i are the edges (j,i), $j \in \mathcal{V}$ and the *outgoing links* of a node i are the edges (i,j), $j \in \mathcal{V}$. The *indegree* of a node i, denoted as $\delta^-(i)$, is the number of incoming links, and similarly the *outdegree* of a node i, denoted as $\delta^+(i)$, is the number of outgoing links. The *maximum indegree* of a graph is $\Delta^- = \max_{i\in\mathcal{V}} \delta^-(i)$ and the *maximum outdegree* of a graph is $\Delta^+ = \max_{i\in\mathcal{V}} \delta^+(i)$. In an undirected graph, the *degree* of a node i is denoted

$\delta(i)$, and $\delta(i) = \delta^-(i) = \delta^+(i)$. The *maximum degree* of an undirected graph is denoted Δ, and $\Delta = \Delta^- = \Delta^+$.

A *complete graph* is an undirected graph where every pair of nodes is adjacent. A *subgraph* of a graph $G = (\mathcal{V}, \mathcal{E})$ is a graph where its vertices are a subset of $\mathcal{V}$ and its edges are a subset of $\mathcal{E}$. The set of nodes $\mathcal{V}' \subset \mathcal{V}$ *induces* a subgraph $G' = (\mathcal{V}', \mathcal{E}')$, where the edges of the induced subgraph $\mathcal{E}' \subset \mathcal{E}$ includes all edges between the nodes in $\mathcal{V}'$. A *clique* in an undirected graph G is a subgraph of G that is a complete graph. The size of a clique is the number of nodes in the clique, and we call a clique with k nodes a *k-clique*.

We may associate a *weight* with the edges on the network, which is represented by a function mapping the set of edges to the set of possible weights. For instance, we can weight the edges by their probability of success, represented by the function $p : \mathcal{E} \mapsto [0, 1]$. We represent the probability of a link $l = (i, j)$ as p_l or p_{ij} instead of $p(l)$ for more compact notation. The resulting *weighted graph* is represented by $G = (\mathcal{V}, \mathcal{E}, p)$ (we use G both for a graph and a weighted graph because the meaning of G should be clear from the context).

A *path* of length n in a graph is a sequence of vertices $(i_1, \ldots, i_{n+1})$ such that an edge exists between each vertex and the next vertex in the sequence. We say two nodes are h *hops* apart if the shortest path between the two nodes has length h. A graph is *connected* if there exists a path between every vertex in the graph. Similarly, a *connected component* (or simply, a *component*) of a graph is a connected subgraph such that no path exists in the original graph between nodes in the subgraph and nodes outside the subgraph.[6] A *cycle* is a path where the first and last vertex in the sequence is the same (in which case any vertex in the cycle can be considered the first or last vertex). A graph is *acyclic* if it does not contain any cycles. We call a *Directed Acyclic Graph* a *DAG*.

A *tree* is a connected, undirected, acyclic graph. We can designate a node in a tree as the *root* node, and call the other nodes in the tree its *descendants*. Let us assume node i has a shorter path to the root node than node j, and an undirected edge exists between i and j. Then, i is the *parent* of node j and j is a *child* of node i. Note that a node in a

[6] Another way to state this is to say that the component is "maximal," meaning it includes all nodes connected to nodes in the component.

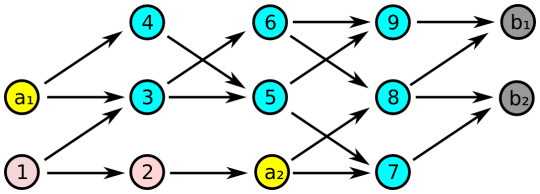

Figure 1.5. Example of a Directed Acyclic Graph (DAG) used to represent a routing topology. The yellow nodes are the sources $\mathcal{A}$, the gray nodes are the sinks $\mathcal{B}$, the cyan nodes are the nodes in the routing topology for routing from $\mathcal{A}$ to $\mathcal{B}$, and the light pink nodes are not in the routing topology. Nodes a_1, 3, and 4 are upstream of node 5 while nodes 7, 9, b_1, and b_2 are downstream of node 5.

tree can only have one parent, but it can have many children. We can construct a *directed tree* from a tree by converting all the undirected edges to directed edges such that they are *all* oriented from a child to a parent in the undirected tree (or they are all oriented from a parent to a child, but not both). Not all connected DAGs are directed trees — in fact, the mesh topologies considered throughout this dissertation are not trees.

In this dissertation, we are interested in DAGs that represent a *routing topology* (to be discussed in Chapter 2). Therefore, we will designate a set of nodes $\mathcal{A}$ as *source nodes* (or a single node a as a source node) and a set of nodes $\mathcal{B}$ as *sink nodes* (or a single node b as a sink node). Typically, $\mathcal{B}$ consists of nodes that only have incoming links. Also, after selecting $\mathcal{A}$, we will only look at the nodes and edges in the DAG that lie on the paths between nodes in $\mathcal{A}$ and nodes in $\mathcal{B}$. On this subgraph of the DAG, $\mathcal{A}$ only has outgoing links. We say node i is *upstream* of node j (and node j is *downstream* of node i) if there exists a path consisting of directed edges from i to j in this subgraph of the DAG. See Figure 1.5 for an illustration of these terms for describing DAG routing topologies.

Chapter 2

Metrics for Constructing Routing Topologies

In this chapter, we will study how to construct a *routing topology* given an *ad-hoc* deployment of wireless nodes. An ad-hoc deployment means that node placement was not planned precisely, unlike the site surveys used to help plan the deployment of cellphone basestations. Initially, each node in the network can determine its *neighbors*, nodes in the network within direct transmission range. Thus, the network is represented as an undirected graph of links, which we call the *connectivity graph*. The routing topology assigns an orientation to the edges in the connectivity graph indicating the paths a packet can follow from its *source* to its *destination* (also called a *sink*). The resulting routing topology is a *directed acyclic graph* (DAG) with multiple paths from the source to the sink.

Our goal is to generate a routing topology such that routing a packet from the source to the sink is likely to succeed. Figure 2.1 gives an example of the choices that need to be made when constructing a routing topology. To attain this goal, we develop a set of *metrics* to measure the reliability of the connection between the source and the sink on a routing topology under different *link failure models* and *routing models*. The "reliability of a connection", which we sometimes call the *end-to-end connectivity*, is the probability that a packet sent from the source is received by the sink (Note that the *actual* end-to-end packet

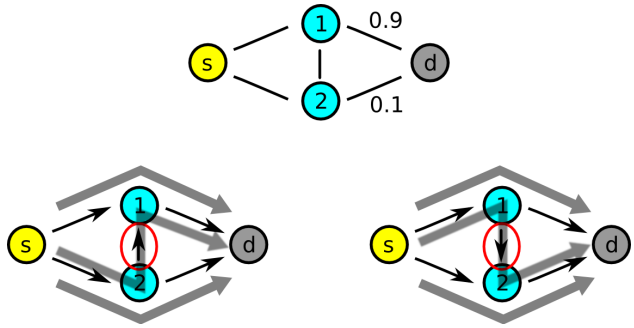

Figure 2.1. Example of constructing a routing topology (bottom, left and right) from a connectivity graph (top). Two of the links in the connectivity graph are labeled with their respective link probabilities. There are two reasonable choices for a routing topology, differing in the orientation of the link between nodes 1 and 2. The topology on the left is a better choice because more paths pass through link $(1, d)$ than link $(2, d)$. Link $(1, d)$ has a higher probability of successfully transmitting the packet to the sink.

delivery probability, which will be covered in Chapter 4, also depends on the schedule used on the network). Using these routing topology metrics a network designer can estimate whether the deployed network is reliable enough for his application. If not, he may place additional relay nodes to add more links and paths to the routing topology. He may also use these metrics to quickly compare different routing topologies and develop an intuition of which ad-hoc placement strategies generate good connectivity graphs.

Section 2.1 will provide some general background on single-path and multi-path routing, with a focus on metrics used to construct routing paths. Then, sections 2.1 and 2.3 will present some new routing topology metrics, followed by a routing topology construction algorithm in Section 2.4.

2.1 Background and Related Work on Routing

The initial work on routing in *wireless ad-hoc networks* focused on route discovery and maintenance for single-path routing. Wireless ad-hoc networks have intermittent connectivity and, unlike *managed (infrastructure) networks* which designate certain nodes as network traffic coordinators (access points, routers) during the physical deployment of the network, all or most nodes can route traffic. Networks *self-organize* into a routing topology, which may change over time depending on the connectivity of the network.

2.1.1 Constructing Single-Path Routes

Many single-path routing algorithms for wireless sensor networks find minimum weight paths using a shortest path algorithm such as Dijkstra's algorithm or the distributed Bellman-Ford algorithm [20], where the *path weight* (sometimes called the *path cost*) is a sum of the *link costs* along the path. Protocols such as MintRoute [113] and Drain [107] (as well as Zigbee, mentioned in Section 1.3.3) differ in the methods employed to calculate the link cost. For instance, MintRoute uses WMEWMA (Windowed Mean Exponentially Weighted Moving Average), defined as

$$WMEWMA(t, \alpha) = \sum_{i=1}^{W} \alpha^i \frac{n_{\text{recv},t}}{\max(n_{\text{expect},t}, n_{\text{recv},t})}$$

where $\alpha \in [0, 1]$ is a tuning parameter, t is the time window, $n_{\text{recv},t}$ is the number of packets received in t, $n_{\text{expect},t}$ is the number of packets expected in t, and W is the moving average history length. Drain uses the inverse of the LQI or RSSI value provided by the 802.15.4 radio as the link cost. Other common link costs used in *802.11* wireless mesh networks include ETX (Expected Transmission Count), ETT (Expected Transmission Time), and RTT (Round Trip Time) [12; 88]. Some link costs, such as mETX (modified Expected Transmission Count) and ENT (Effective Number of Transmissions) [52], try to also incorporate link variability (via the standard deviation) into the cost.

While the link costs mentioned above are summed to get a path cost, there are other ways to compute the path cost from link costs. For instance, ML (Minimum Loss) [12] and SPP (Success Probability Product) [88] (two names for the same path cost) both multiply link probabilities to get a path cost. Weighted Cumulative Expected Transmission Time (WCETT) [26], designed for balancing channel usage on multi-radio wireless mesh networks, is a path cost which uses a combination of summation and the max operator on ETT:

$$WCETT = (1 - \beta) \sum_{i=1}^{n} ETT_i + \beta \max_{1 \le j \le k} X_j$$

$$X_j = \sum_{\substack{\text{hop } i \text{ is on} \\ \text{channel } j}} ETT_i$$

where $\beta \in [0, 1]$ is a tunable parameter, n is the number of links on the path, and k is the number of channels. WCETT is not suitable for finding minimum cost paths in a distributed

fashion. In fact, an important consideration for selecting a path cost is whether finding a minimum cost path requires global link cost information.

In single-path routing, when a link fails for multiple consecutive packets (the packet reception rate on the link falls below a threshold), the paths using that link are invalidated and replaced by new paths. The process of building new paths can take a significant amount of time, bandwidth, and computing resources. Worst of all, the wireless connectivity may change by the time the new paths are built.

2.1.2 Constructing Multi-Path Routes

Alternatively, multi-path routing from a single source to a single sink can reduce the number of times a path needs to be rebuilt, while balancing the load on the network and improving the end-to-end reliability between the source and the sink. There are many types of multi-path routing.

- *Multi-path Source Routing* (e.g., Braided Multi-path [34]) sends packets along multiple paths, specifying the entire routing path from the source to the sink for each packet.
- *Mesh Routing* (e.g., Time Synchronized Mesh Protocol [80]) does not specify an end-to-end path for each packet, but instead sets up a "mesh of paths" (multiple paths that share links) for the packets that all flow to the sink. A node in the mesh tries to transmit a packet on its outgoing links in a deterministic order.
- Finally, *Flooding* or some form of *Constrained Flooding* (e.g., GRAdient Broadcast [116], Trickle [59]) sends multiple copies of a packet down multiple paths to the sink (or to all nodes). Some schemes, such as the scheme described by Dulman et al. [28], code the data over a set of packets and send them along disjoint or braided (partially disjoint) paths such that only a subset of the packets needs to be received to reconstruct the data.[1]

[1] Strictly speaking, this would not be considered a flooding scheme.

The stochastic counterpart to all these multi-path routing schemes select links / paths for packet transmission following a probability distribution. For instance, Constrained Random Walk [94] is an example of *Stochastic Mesh Routing* while ARRIVE [48] is an example of *Stochastic Constrained Flooding* (ARRIVE also duplicates packets following a probability distribution).

Note that even in deterministic routing schemes, the actual path taken by a packet may not be "deterministic," or known apriori, because of the stochastic nature of link failures. The stochastic nature of wireless links also affects the path of packets in *Opportunistic Routing* schemes like ExOR [9]. Opportunistic routing takes advantage of overhearing transmissions and acknowledgments from neighbors to coordinate and select, on the fly, which nodes should relay packets. If the wireless network channels were not stochastic, the paths of packets would be deterministic.

Some multi-path routing algorithms do not use link or path costs to help select links for the routing topology (some single-path algorithms don't either). For instance, in multi-path source routing protocols such as SMR (Split Multipath Routing) [58], AOMDV (Ad-hoc On-demand Multipath Distance Vector) [70], and AODVM (Ad-hoc On-demand Distance Vector Multipath) [118], the focus is on finding *link-disjoint* or *node-disjoint* paths (paths which do not share links or nodes) from the source to the sink. SMR chooses two paths, the path with the shortest delay and the next path that is maximally disjoint from the first path. In case of a tie for the second path, it chooses the one with the shortest hop count and delay. Similarly, AOMDV and AODVM find link-disjoint and node-disjoint paths that respond first during the route discovery process or have the shortest hop count. Path costs can be incorporated into the path selection criteria by looking for disjoint paths with the minimum aggregate path cost (sum of all path costs). For instance, Bhandari [8] gives two algorithms to find the minimum aggregate cost link-disjoint and node-disjoint paths, where an individual path cost is a sum of the link costs.

In mesh routing and constrained flooding protocols, much of the literature either does not use link costs or uses a sum of link costs to help construct the routing topology. In M-MPR (Meshed Multipath Routing) [23], the geographic location of the nodes helps the

initial discovery of routing paths. Nodes are restricted to two forwarding links to reduce the control overhead of maintaining routes, and the first two routes found are selected for the routing topology. In GRAB (GRAdient Broadcast) [116], link costs are added to assign a cost to intermediate routing nodes, which then forms a cost potential field for directing packets to the sink.

On the other hand, Dubois-Ferriere [27] uses link costs in a similar fashion to the use of link costs in this dissertation to construct routing topologies. Dubois-Ferriere proposed Anypath Routing, a class of multi-path opportunistic routing protocols. ERS-best[2] Anypath Routing is a mesh routing protocol whereby a node i transmits a packet to a set of "next-hop" nodes $\mathcal{C}_i$, and one of the nodes (the one with the minimum node cost) who receives the packet continues to route the packet forward. The set $\mathcal{C}_i$ is selected from the neighbors of i, $\mathcal{N}_i$, to minimize the cost at node i, W_i. A Bellman-Ford type of algorithm is described in [27] to simultaneously find $\mathcal{C}_i$ and compute W_i on a DAG. W_i is related to $\mathcal{C}_i$ by the set of recursive equations

$$W_i = w_{i\mathcal{C}_i} + R_{i\mathcal{C}_i}, \quad W_b = 0 \tag{2.1}$$

$$w_{i\mathcal{C}_i} = \frac{1}{1 - \prod_{j \in \mathcal{C}_i}(1 - p_{ij})} \tag{2.2}$$

$$R_{i\mathcal{C}_i} = p_{ij_1} W_{j_1} + \sum_{h=2}^{|\mathcal{C}_i|} \left(\prod_{k=1}^{h-1} (1 - p_{ij_k}) \right) p_{ij_h} W_{j_h} \tag{2.3}$$

where p_{ij} is the probability of link (i, j), b is the destination node id, and for convenience of notation we assume that the nodes in $\mathcal{N}_i$ have node ids $j_1 < j_2 < \cdots < j_{|\mathcal{C}_i|}$ assigned such that $W_{j_1} < W_{j_2} < \cdots < W_{j_{|\mathcal{C}_i|}}$. Here, we chose a particular ETX inspired *anypath link cost*, $w_{i\mathcal{C}_i}$, from the possible anypath link costs described in [27].[3] $R_{i\mathcal{C}_i}$ stands for the remaining path cost, which is the expected cost assuming the node with the lowest id who received the packet is chosen to forward the packet. ERS-all Anypath Routing is a constrained flooding protocol where all nodes in $\mathcal{C}_i$ who receive a copy of the packet forwards the packet. The set of next-hop nodes $\mathcal{C}_i$ minimizes the cost W_i, which can be computed using (2.1), (2.2),

[2] ERS stands for Effective Relay Selection.

[3] Note that an anypath "link cost" is between a node i and a set $\mathcal{C}_i$, not just between two nodes i and j.

and

$$R_{i\mathcal{C}_i} = \sum_{j \in \mathcal{C}_i} W_j \qquad . \tag{2.4}$$

2.1.3 Analysis of Multi-Path Routing

Many papers on multi-path routing use simulations to demonstrate qualitative features of their routing schemes [34; 48]. A small set of papers try to mathematically model and analyze the benefits of multi-path routing, but they either model at the level of paths [28; 75] or assume the networks have a very large number of nodes [35]. In [28], Dulman et al. perform some simple analysis to get the tradeoff between traffic and reliability, but the analysis does not consider latency. Furthermore, the calculations use the end-to-end connection probability of disjoint *paths*, not individual link probabilities, and hence do not account for varying path lengths or link probabilities. In [75], Nasipuri et al. propose a multi-path extension to DSR (Dynamic Source Routing) and the analysis focuses on finding the statistics of the time between successive route discoveries. Again, the paper builds on a path model with path lifetimes drawn from a distribution instead of a link model with individual link probabilities. In [35], the authors use a geometry-based argument on networks with a very large number of nodes to argue that k-shortest path routing algorithms, which find the k shortest disjoint paths, only distribute the load evenly through a network when it uses a very large number of paths.

Of particular interest is the modeling and analysis of SERAN [10] and Breath [78], two cluster-based routing protocols designed for industrial control over WSN. Cluster-based routing is a form of constrained flooding, where multiple copies of a packet are passed between groups of nodes to get higher reliability. SERAN and Breath assume the independence of links, node wake up times, and random attempts to access the channel so that the Central Limit Theorem can be employed to get probabilistic guarantees on end-to-end latency and end-to-end reliability.

In Chapter 4, we will derive models and analysis tools for *Unicast Path Diversity*, a class of TDMA mesh routing protocols, and *Directed Staged Flooding*, a class of TDMA

constrained flooding protocols, using link probabilities collected from the network. The models do not only apply to large networks, and does not employ the Central Limit Theorem to approximate delay, although links *are* assumed to be independent over time. The models are presented in Chapter 4 because they only apply *after* the routing topologies and schedules are constructed.

2.2 Flooding Connectivity Metrics

This section presents two routing topology metrics, the *path probability metric* and the *robustness metric*. Both metrics fall under the class of *flooding connectivity metrics*, which are metrics that assume multiple copies of a packet are routed on all the paths in the network. On the other hand, Section 2.3 will present metrics that assume only one copy of the packet is routed through the network. The robustness metric is an approximation of the path probability metric that is easier to compute. This will be explained in more detail in Section 2.2.3. We will use these metrics to show that multiple interleaved paths typically provide better end-to-end connectivity than disjoint paths.

2.2.1 Routing and Link Failure Models

Both the path probability metric and the robustness metric share the same link failure model. We assume that the links in the network succeed and fail independently of each other. In reality, link failure is affected by the quality of the radios in the network, external interference sources, and multi-path fading from physical obstacles in the environment. We also assume that each node can estimate the probability that an incoming or outgoing link fails, perhaps through link estimation techniques at the physical layer such as the ones described for 802.15.4 in Section 1.3.3.

The routing model for both flooding connectivity metrics is that after a node receives a packet, it multicasts the packet *once* on all its outgoing links. The node multicasts the packet only after hearing from all its upstream neighbors, so there are no retransmissions on a link even if the node receives multiple copies of the packet. In this manner, copies

of a packet will attempt to traverse all possible paths through the network. Note that the primary difference between this routing model and general flooding on a network is that the multicast must respect the orientation of the edges in the routing topology DAG. Also, the routing topology restricts the order in which nodes can multicast, unlike general flooding where nodes broadcast to their neighbors in an ad-hoc order.

Given our routing and link failure models, we can represent the routing topology as a weighted DAG $G = (\mathcal{V}, \mathcal{E}, p)$ where each link l is weighted by its link success probability p_l (only one probability is associated with each link and the link probabilities are independent). The routing metrics are computed from G and the source-destination pair (a, b).

2.2.2 Path Probability Metric

Definition 2.2.1 (Path Probability Metric)**.** Let $G = (\mathcal{V}, \mathcal{E}, p)$ be a weighted DAG where all nodes have an outgoing edge except for the destination node b, and let p_{ij} represent the probability of link (i, j). The *path probability* metric, or simply the *path probability*, $p_{a\to b} \in [0, 1]$ is the probability that a path exists in G between the source a and the sink b. When there are several graphs, we use the notation $p^G_{a\to b}$ to specify $p_{a\to b}$ on graph G. □

Algorithm 1 calculates $p_{a\to b}$, by enumerating all possible paths. Unfortunately, it is very computationally expensive, taking $O(|\mathcal{E}| \cdot 2^{|\mathcal{E}|})$ to compute where $|\mathcal{E}|$ is the number of edges.

Alternatively, Algorithm 2 computes the path probability between a and b using dynamic programming and is significantly faster. The state used by the dynamic programming algorithm is the joint probability distribution of receiving a packet on *vertex cuts* of the graph separating a and b. A vertex cut of a and b on a connected graph is a set of nodes $\mathcal{C}$ such that the subgraph induced by $\mathcal{V}\backslash\mathcal{C}$ does not have a single component that contains both a and b. This definition allows a and b to be elements of $\mathcal{C}$, which is necessary for the first and last steps of the algorithm. During most steps of the algorithm, $\mathcal{C}$ is chosen such that removing $\mathcal{C}$ partitions the graph into two disconnected components, one containing a and the other containing b. Let $\mathcal{C}^{(k)}$ denote the vertex cut chosen in step k of the algorithm.

Algorithm 1 PATH_PROB (Path Probability Algorithm)

Input: $G = (\mathcal{V}, \mathcal{E}, p), a, b$

Output: $p_{a \to b}$

$p_{a \to b} := 0$

for all $\mathcal{E}' \in 2^{\mathcal{E}}$ **do** ▷ $\mathcal{E}'$ is a subset of $\mathcal{E}$.

  if $\mathcal{E}'$ contains a path connecting a and b **then**

    $p_{a \to b} := p_{a \to b} + \prod_{l \in \mathcal{E}'} p_l \prod_{\bar{l} \in \mathcal{E} \setminus \mathcal{E}'} (1 - p_{\bar{l}})$

  end if

end for

Return: $p_{a \to b}$

Conceptually, the algorithm is converting the DAG representing the network to a *vertex cut DAG*, where each vertex cut $\mathcal{C}^{(k)}$ is represented by the set of nodes $\mathcal{S}^{(k)} = 2^{\mathcal{C}^{(k)}}$. Each node in $\mathcal{S}^{(k)}$ represents the event that a particular subset of the vertex cut received a copy of the packet. The algorithm computes a probability for each node in $\mathcal{S}^{(k)}$, and the collection of probabilities of all the nodes in $\mathcal{S}^{(k)}$ represent the joint probability distribution that nodes in the vertex cut $\mathcal{C}^{(k)}$ can receive a copy of the packet. A link in the vertex cut DAG represents a valid (nonzero probability) transition from a subset of nodes that have received a copy of the packet in $\mathcal{C}^{(k-1)}$ to a subset of nodes that have received a copy of the packet in $\mathcal{C}^{(k)}$. Figure 2.3 shows an example of this graph conversion using the selection of vertex cuts depicted in Figure 2.2.

Algorithm 2 invokes Procedure 3 to find an ordering (represented by the queue $\mathcal{Q}$) for adding nodes to the vertex cut such that the vertex cut stays small throughout the execution of the algorithm. A node can only be added to the vertex cut if all its incoming links originate from the vertex cut. When a node is added to the vertex cut, its incoming links are removed. A node is removed from the vertex cut if all it's outgoing links have been removed. Note that Procedure 3 is a greedy algorithm and may not always find an ordering that minimizes the size of the largest vertex cut, $\hat{C}$, used during the execution of the algorithm.

Algorithm 2 PATH_PROB2 (Optimized Path Probability Algorithm)

Input: $G = (\mathcal{V}, \mathcal{E}, p), a$ ▷ G is a connected DAG.

Output: $\{p_{a\to v}, \forall v \in \mathcal{V}\}$

$(\mathcal{Q}, \hat{C}) :=$ ADD_NODE_LIST(G, a) ▷ $\mathcal{Q}$ is a queue. $\hat{C}$ not used here.

$\mathcal{C} := \{a\}$ ▷ $\mathcal{C}$ is the vertex cut.

$\mathcal{E}' := \mathcal{E}$ ▷ $\mathcal{E}'$ is the set of remaining edges.

$p_{\mathcal{C}}(\{a\}) := 1$, $p_{\mathcal{C}}(\emptyset) := 0$ ▷ $p_{\mathcal{C}}(\mathcal{C}')$ is the probability $\mathcal{C}' \subseteq \mathcal{C}$ all have packets.

while $\mathcal{Q} \neq \emptyset$ **do**

[Add node to vertex cut]

$v :=$ dequeue$(\mathcal{Q})$

$p'_{\mathcal{C}} :=$ `NIL` ▷ Probabilities for next vertex cut $\mathcal{C} \cup \{v\}$. `NIL` means not yet assigned.

for all $\mathcal{C}' \in 2^{\mathcal{C}}$ **do** ▷ $\mathcal{C}'$ is a subset of $\mathcal{C}$.

Let $\mathcal{L} = \{(u, v) \in \mathcal{E}' \,:\, u \in \mathcal{C}'\}$ ▷ v's incoming links from $\mathcal{C}'$.

$p'_{\mathcal{C}}(\mathcal{C}' \cup \{v\}) := p_{\mathcal{C}}(\mathcal{C}') \cdot \left(1 - \prod_{l\in\mathcal{L}}(1 - p_l)\right)$

$p'_{\mathcal{C}}(\mathcal{C}') := p_{\mathcal{C}}(\mathcal{C}') \cdot \prod_{l\in\mathcal{L}}(1 - p_l)$

end for

$\mathcal{E}' := \mathcal{E}' \backslash \{(u, v) \in \mathcal{E}' \,:\, u \in \mathcal{C}\}$ ▷ Remove all of v's incoming links.

$\mathcal{C} := \mathcal{C} \cup \{v\}$

[Compute path probability]

Let $2^{\mathcal{C}}_v = \{\mathcal{C}' \in 2^{\mathcal{C}} \,:\, v \in \mathcal{C}'\}$ ▷ Subsets of $\mathcal{C}$ that contain node v.

$p_{a\to v} := \sum_{\mathcal{C}'_v \in 2^{\mathcal{C}}_v} p'_{\mathcal{C}}(\mathcal{C}'_v)$

[Remove nodes from vertex cut]

Let $\mathcal{D} = \{i \in \mathcal{C} \,:\, \forall j,\, (i, j) \notin \mathcal{E}'\}$ ▷ Nodes with no outgoing links.

$\mathcal{C} := \mathcal{C} \backslash \mathcal{D}$

$p_{\mathcal{C}} :=$ `NIL`

for all $\mathcal{C}' \in 2^{\mathcal{C}}$ **do** ▷ Combine probabilities for removed nodes.

$p_{\mathcal{C}}(\mathcal{C}') := \sum_{\mathcal{D}' \in 2^{\mathcal{D}}} p'_{\mathcal{C}}(\mathcal{C}' \cup \mathcal{D}')$

end for

end while

Return: $\{p_{a\to v}, \forall v \in \mathcal{V}\}$

Procedure 3 ADD_NODE_LIST

Input: $G = (\mathcal{V}, \mathcal{E}, p), a$ ▷ G is a connected DAG.

Output: $\mathcal{Q}, \hat{C}$ ▷ $\mathcal{Q}$ is a queue. $\hat{C}$ is the size of the largest vertex cut.

$\mathcal{Q} := (a)$, $\hat{C} := 1$

$\mathcal{C} := \{a\}$ ▷ $\mathcal{C}$ is the vertex cut.

$\mathcal{V}' := \mathcal{V} \backslash a$ ▷ $\mathcal{V}'$ is the set of remaining vertices.

$\mathcal{E}' := \mathcal{E}$ ▷ $\mathcal{E}'$ is the set of remaining links.

$u := a$ ▷ u is the node targeted for removal from $\mathcal{C}$.

while $\mathcal{V}' \neq \emptyset$ **do**

[Find next node u to remove from vertex cut]

if $u \notin \mathcal{C}$ **then**

Let $\mathcal{J} = \{j : \forall (i,j) \in \mathcal{E}', i \in \mathcal{C}\}$ ▷ Nodes with all incoming links from $\mathcal{C}$.

$u := \arg\min_{i \in \mathcal{C}} \left|\{(i,j) \in \mathcal{E}' : j \in \mathcal{J}\}\right|$ ▷ Node with fewest outgoing links to $\mathcal{J}$.

end if

[Add node(s) to / Remove node(s) from vertex cut]

Select a node $v \in \{v \in \mathcal{V} : (u,v) \in \mathcal{E}'\}$ ▷ v is a child of u.

enqueue$(\mathcal{Q}, v)$

$\mathcal{C} := \mathcal{C} \cup \{v\}$

$\hat{C} := \max(\hat{C}, |\mathcal{C}|)$

$\mathcal{V}' := \mathcal{V}' \backslash v$

$\mathcal{E}' := \mathcal{E}' \backslash \{(i,v) \in \mathcal{E}' : i \in \mathcal{C}\}$ ▷ Remove all of v's incoming links.

$\mathcal{C} := \mathcal{C} \backslash \{i \in \mathcal{C} : \forall j, (i,j) \notin \mathcal{E}'\}$ ▷ Remove nodes with no outgoing links.

end while

Return: $\mathcal{Q}, \hat{C}$

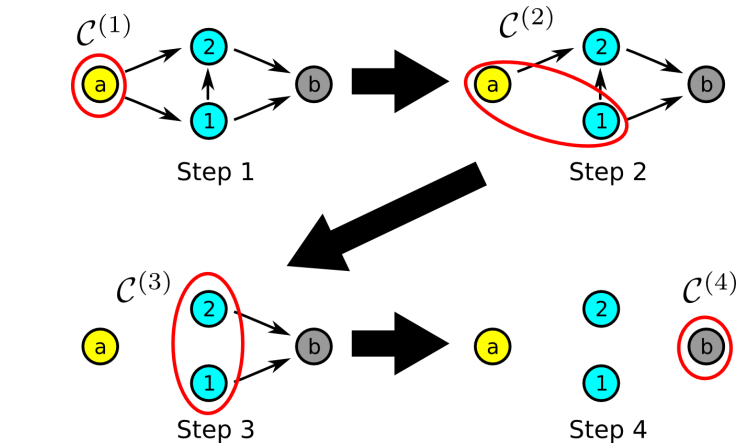

Figure 2.2. An example of a sequence of vertex cuts that can be used by Algorithm 2. The vertex cut after adding and removing nodes from each iteration of the outer loop is circled in red.

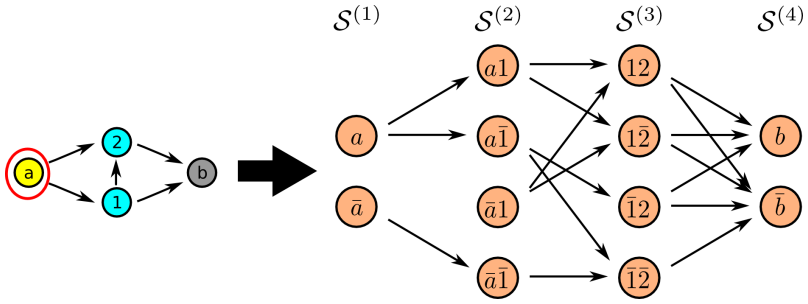

Figure 2.3. Running Algorithm 2 on the network graph shown on the left when selecting vertex cuts in the order depicted in Figure 2.2 is equivalent to creating the vertex cut DAG shown on the right and finding the probability that state a will transition to state b. Each column of nodes $\mathcal{S}$ in the vertex cut DAG represents the joint probability distribution of receiving the packet over a vertex cut in the network.

Computing the path probability $p_{a\to b}$ reduces to computing the joint probability distribution that a packet is received by a subset of the vertex cut in each step of the algorithm. The joint probability distribution over the vertex cut $\mathcal{C}^{(k)}$ is represented by the function $p_{\mathcal{C}}^{(k)} : \mathcal{S}^{(k)} \mapsto [0,1]$. Step k of the algorithm computes $p_{\mathcal{C}}^{(k)}$ from $p_{\mathcal{C}}^{(k-1)}$ by lines 13, 14, and 26 in Algorithm 2. Notice that the nodes in each $\mathcal{S}^{(k)}$ represent disjoint events, which is why we can combine probabilities in line 26 using summation.

The running time of Algorithm 2 depends on the size of the largest vertex cut used in the algorithm, $\hat{C}$, computed by Procedure 3.[4] The running time of the operations in lines 11–12 of Procedure 3 is $O(\hat{C}\Delta^+)$, where $\Delta^+ = \max_{v\in\mathcal{V}} \delta^+(v)$ is the maximum outdegree of

[4]It would be preferable to bound the running time by a property of the graph. One candidate is the the max-cut of the graph when edges are weighted by 1, but computing the max-cut of a graph is NP-hard [49].

G. Lines 15–21 have constant running time. Therefore, the running time of Procedure 3 is $O(|\mathcal{V}|\hat{C}\Delta^+)$. The outer loop of Algorithm 2 (lines 7–28) runs $|\mathcal{V}|$ times. The computation in the outer loop is dominated by the loops over the power set of the vertex cuts $\mathcal{C}$ used by the algorithm (lines 11–15, 19–20, 25–27). In the loop on lines 11–15, the worst case running time for each iteration of the loop is linear in the maximum indegree of the vertices, $\Delta^- = \max_{v\in\mathcal{V}} \delta^-(v)$. In the loop on lines 25–27, the number of summation operations times the number of loops is always less than $2^{\hat{C}}$, so the running time is $O(2^{\hat{C}})$.[5] Therefore, the running time of the entire algorithm is $O\big(|\mathcal{V}|(\hat{C}\Delta^+ + 2^{\hat{C}}\Delta^-)\big)$, which is typically much smaller than $O(|\mathcal{E}| \cdot 2^{|\mathcal{E}|})$, the running time of Algorithm 1.

The main drawback with using the path probability as a metric is that it requires global knowledge of the network topology and link probabilities to compute. Even Algorithm 2 requires knowledge of the outgoing link probabilities of a vertex cut of the network. The nodes in a vertex cut may not be in communication range of each other, and thus the communication necessary to run Algorithm 2 in-network will not be local (i.e., between neighboring nodes). The next section proposes a metric that can be computed with only local communication.

2.2.3 Robustness Metric

Definition 2.2.2 (Robustness Metric)**.** Let $G = (\mathcal{V}, \mathcal{E}, p)$ be a weighted DAG where all nodes have an outgoing edge except for the destination node b, and let p_{ij} represent the probability of link (i, j). The *robustness metric* $r_{a\to b} \in [0, 1]$ on G with respect to a source-destination pair (a, b) can be computed using

$$\begin{aligned} r_{a\to a} &= 1 \\ r_{a\to v} &= 1 - \prod_i (1 - r_{a\to u_i} p_{u_i v}) \end{aligned} \tag{2.5}$$

where u_i are the upstream neighbors of node v. When there are several graphs, we use the notation $r^G_{a\to v}$ to specify $r_{a\to v}$ on graph G. □

[5]The summation on lines 19–20 are also $O(2^{\hat{C}})$. Actually, it is not necessary to compute $p_{a\to v}$ for all v if we are only interested in $p_{a\to b}$.

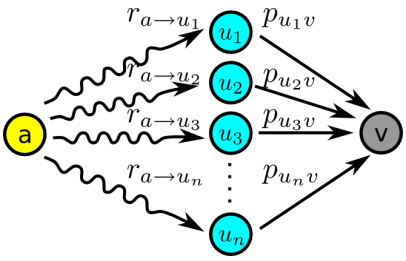

Figure 2.4. The paths from a to each downstream node u_i are treated as if they share no links during the robustness metric calculation.

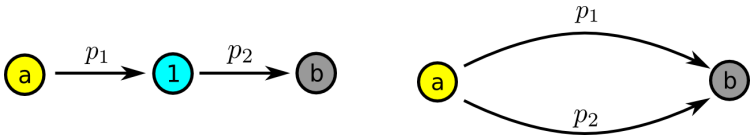

Figure 2.5. (left) When nodes a and b are connected in series, the path probability is just a product of the link probabilities $p_1 p_2$. (right) When nodes a and b are connected in parallel, the path probability is just $1-(1-p_1)(1-p_2)$.

Figure 2.4 illustrates (2.5). The motivation for defining the robustness metric in this fashion is that it captures how to compute the path probability between two nodes when there are independent links in series and in parallel, as illustrated in Figure 2.5.

Note that if we reverse the orientation of all the links on a graph G to get $\overleftarrow{G} = (\mathcal{V}, \overleftarrow{\mathcal{E}}, p)$, where $\overleftarrow{\mathcal{E}} = \{(i,j) : (j,i) \in \mathcal{E}\}$, then $r^{\overleftarrow{G}}_{b\to a}$ does not equal $r^{G}_{a\to b}$ (See Example 2.2.4 in Section 2.2.4).[6] This is due to the nested, recursive nature of (2.5). On the other hand, the path probability $p^{\overleftarrow{G}}_{b\to a}$ is equal to $p^{G}_{a\to b}$.

The definition of the robustness metric naturally leads to a dynamic programming algorithm to compute it in $O(|\mathcal{V}| + |\mathcal{E}|)$ time, Algorithm 4. Algorithm 4 can easily be modified such that it is distributed across all the nodes in the network. A node v simply waits for the robustness metric on all its upstream neighbors nodes to be computed before computing its own robustness metric.

As mentioned earlier, the robustness metric is an approximation of the path probability $p_{a\to b}$ that requires only local information and is faster to compute. Example 2.2.2 in Section 2.2.4 gives a routing topology where $r_{a\to b} \neq p_{a\to b}$. The robustness metric *would* equal the path probability on topologies where all the paths are independent. There is an

[6] Here we break with the convention where a is the source and b is the sink; Instead, b is the source of $\overleftarrow{G}$ while a is the sink of $\overleftarrow{G}$.

Algorithm 4 ROBUST_METRIC

Input: $G = (\mathcal{V}, \mathcal{E}, p), a, b$

Output: $r_{a\to b}$

$\forall v \in \mathcal{V}\backslash a, \quad r_{a\to v} := \texttt{NIL}$ ▷ `NIL` means not yet assigned.

$r_{a\to a} := 1$

$\mathcal{Q} := \{v : (a, v) \in \mathcal{E}\}$ ▷ $\mathcal{Q}$ is a queue.

while $\mathcal{Q} \neq \emptyset$ **do**

  $v := \text{dequeue}(\mathcal{Q})$

  if $\forall u_i \in \{u : (u, v) \in \mathcal{E}\}, r_{a\to u_i} \neq \texttt{NIL}$ **then**

    $r_{a\to v} := 1 - \prod_i (1 - r_{a\to u_i} p_{u_i v})$

    for all $w \in \{w \in \mathcal{V} : (v, w) \in \mathcal{E}\}$ **do**

      if $w \notin \mathcal{Q}$ **then**

        $\text{enqueue}(\mathcal{Q}, w)$

      end if

    end for

  else

    $\text{enqueue}(\mathcal{Q}, v)$

  end if

end while

Return: $r_{a\to b}$

implicit assumption in (2.5) that $r_{a\to u_i}$ is the probability of the event $A_{a\rightsquigarrow u_i}$ that a path exists between a and u_i, and that for all u_i, $A_{a\rightsquigarrow u_i}$ are independent events. This leads us to Conjecture 2.2.1.

Conjecture 2.2.1 ($r_{a\to b} \geq p_{a\to b}$)**.** *For any DAG $G = (\mathcal{V}, \mathcal{E}, p)$ with source a and destination b*

$$r_{a\to b} \geq p_{a\to b} \quad . \tag{2.6}$$

□

Conjecture 2.2.1 states that the robustness metric is an upper bound on the path probability. The intuition behind Conjecture 2.2.1 comes from the following argument. First, we construct a graph $G' = (\mathcal{V}', \mathcal{E}')$ from graph G such that the path probability $p_{a\to b}$ on G' is equal to the robustness metric $r_{a\to b}$ on G. Recall (2.5), where for each $r_{a\to v}$, we treat the paths from a to v through each upstream neighbor u_i as if they were parallel and did not share any links (See Figure 2.4). We construct G' from G by making a duplicate of the intermediate nodes and links between a and b using the following procedure.

Let $G'_{a\rightsquigarrow u_i}$ be graphs constructed thus far and $G'_{a\rightsquigarrow v}$ be the graph we are constructing in the current step. First, set $G'_{a\rightsquigarrow a} = (\{a\}, \emptyset)$. We will select the nodes $v \in \mathcal{V}$ to construct G' in the same order that nodes are taken out of the queue $\mathcal{Q}$ in Algorithm 4. Choose a node v for the current step. For each upstream neighbor u_i of v, make a copy of all the nodes and links in $G'_{a\rightsquigarrow u_i}$ except a, which will be shared by all the graphs. Connect these graphs $G'_{a\rightsquigarrow u_i}$ with the links from u_i to v to form $G'_{a\rightsquigarrow v}$. Choose the next node v following the order in Algorithm 4 and repeat until we have $G'_{a\rightsquigarrow b}$. Finally, set $G' = G'_{a\rightsquigarrow b}$ and for clarity relabel a to a' and b to b'. An example of this is illustrated in Figure 2.6. By construction, $p_{a'\to b'}$ on G' is equal to $r_{a\to b}$ on G. Note that the final graph is not equivalent to a graph where all paths from the source to the sink are parallel.

Now, it remains to show that $p_{a'\to b'} \geq p_{a\to b}$. The intuition is that the paths in G' share less links and are "more independent" than the paths in G. Therefore, we expect that the probability that at least one of the paths in G' succeeds is larger than the probability that at least one of the paths in G succeeds. We expect that if our link failure model for G' is

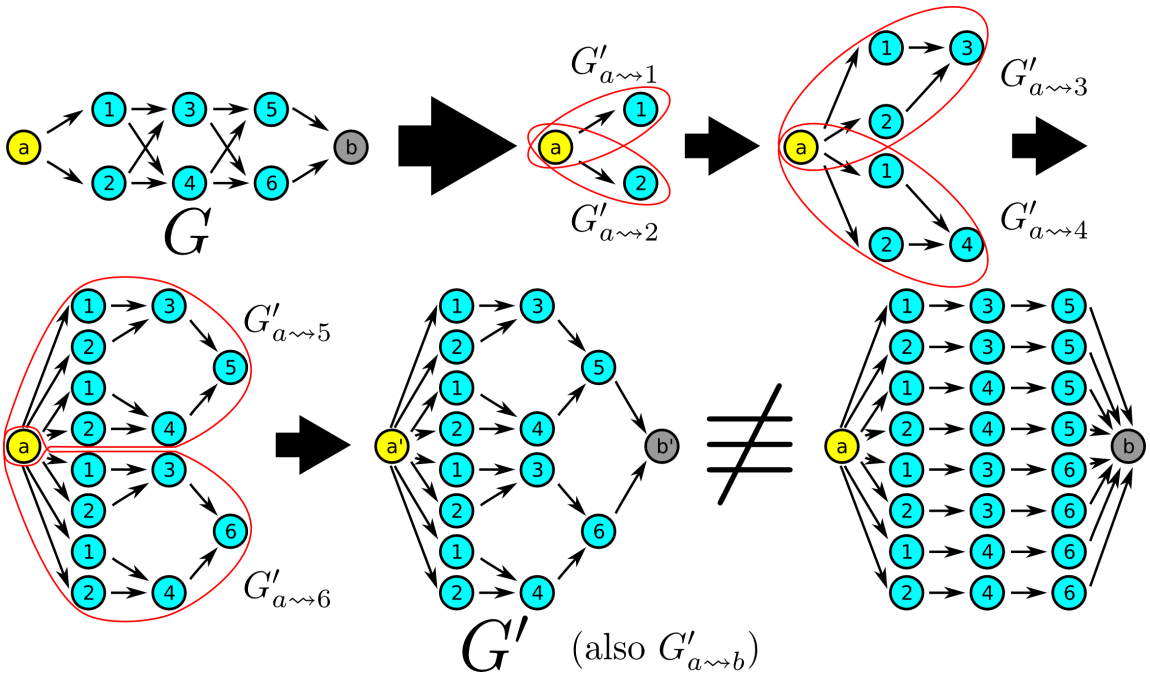

Figure 2.6. Example of steps to convert graph G to graph G'. Note that the final graph is not equivalent to a graph where all paths from the source to the sink are parallel. See text for details.

modified such that the links in each set of duplicated links (links that correspond to one link in G) fail or succeed together, the path probability on G' would be the same as the path probability on G.

To see why we expect $p_{a'\to b'} \geq p_{a\to b}$, let us consider only two paths. Note that each path in G is represented in G' by a path with the same link probabilities. Furthermore, for any pair of paths (g_1, g_2) in G, the corresponding pair of paths (g'_1, g'_2) in G' either shares the same number of links or shares less links. Let A_1 represent the event that path g_1 is successful. Define A_2, A'_1, and A'_2 in a similar fashion for their respective paths. Let $\mathcal{E}_1$ be the set of links in path g_1, and define $\mathcal{E}_2, \mathcal{E}'_1$, and $\mathcal{E}'_2$ in a similar fashion. Then,

$$\prod_{i\in\mathcal{E}_1\cup\mathcal{E}_2} p_i = P(A_1 \cap A_2) \quad \geq \quad P(A'_1 \cap A'_2) = \prod_{j\in\mathcal{E}'_1\cup\mathcal{E}'_2} p_j$$

because we can map each link in $(\mathcal{E}_1 \cup \mathcal{E}_2)$ to a link in $(\mathcal{E}'_1 \cup \mathcal{E}'_2)$ with the same probability and still have leftover links in $(\mathcal{E}'_1 \cup \mathcal{E}'_2)$ (we have duplicated links in G'). This means that

$$P(A_1 \cup A_2) \leq P(A'_1 \cup A'_2)$$

because $P(A_1) = P(A'_1)$ and $P(A_2) = P(A'_2)$ (and in general, $P(A \cup B) = P(A) + P(B) -$

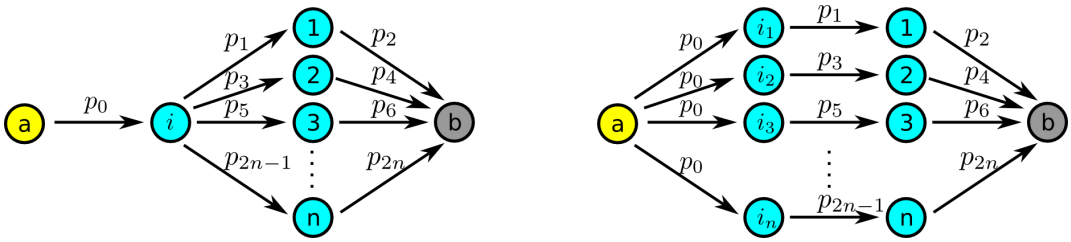

Figure 2.7. The two topologies have the same robustness metric. Links are labeled by their probabilities. Here, the probability subscripts do not correspond to links; Instead, they are used to denote which links in the two graphs have the same probability. Any probabilities with matching subscripts have the same value. Link (a, i) on the left topology, is treated as multiple independent links in the robustness metric calculation.

$P(A \cap B)$). It is more likely that at least one of g_1' and g_2' succeeds than at least one of g_1 and g_2 succeeds. □

Unfortunately, there exists a graph G such that the difference between the robustness metric and the path probability is almost 1, i.e., $\sup_G |r^G_{a\to b} - p^G_{a\to b}| = 1$. For instance, we can construct topologies consisting of a link (a, i) in series with many parallel links, as shown on the left of Figure 2.7. The path probability must be less than p_0, the probability of link (a, i), which we can set to be close to 0. But by adding more parallel links after link (a, i), we can make the robustness metric between the source and sink arbitrarily close to 1. We will see an example of such a topology in the next section.

Although there exists graphs where the robustness metric is not a good approximation of the path probability, in practice the robustness metric is still a useful estimate of the end-to-end connectivity of a routing topology. The robustness metric has the following desirable properties:

- The value of the metric lies between 0 and 1, roughly corresponding to the probability that a path exists between the source and the sink.
- When the metric equals 0, no path exists between the source and the sink.
- Adding a link to a graph does not decrease the value of the metric on the graph.
- Adding a link not on the path between the source and the sink does not change the robustness metric.

- If we fix the sink b, the metric measured from a source a_1 *may* be larger than the metric measured from a source a_2 that is on a path between a_1 and b when a_1 has other paths to reach the sink.
- The metric can be computed efficiently with little communication between the nodes.

We will use a variant of the robustness metric to construct a routing topology from a connectivity graph in Section 2.4.

2.2.4 Examples and Discussion

In this section, we will examine the difference between the robustness metric and the path probability metric on several special examples of graph topologies.

Example 2.2.1 (Serial-parallel topology). Figure 2.8 gives an example of a topology where the robustness metric at the sink is much larger than the path probability at the sink. In fact, $r_{a\to b} = 0.878$ is much greater than $r_{a\to v} = 0.7$ of the downstream neighbor of the source, even though all paths from the source to the sink must pass through the source node's downstream neighbor.

□

The next two examples demonstrate that having many interleaved paths in the mesh can significantly improve connectivity.

Example 2.2.2 (Meshed paths vs. disjoint paths). Figure 2.9 depicts a mesh topology with many interleaved paths and a topology with parallel, disjoint paths. Both topologies have links with the same probability, and both topologies have the same number of links between the columns of nodes in the middle of the graph. However, $p_{a\to b}$ of the meshed paths topology is greater than $p_{a\to b}$ of the disjoint paths topology.

□

Example 2.2.3 (Wide path topologies). This example studies *wide path topologies*, which are routing topologies where nodes can be grouped into columns and links are only between

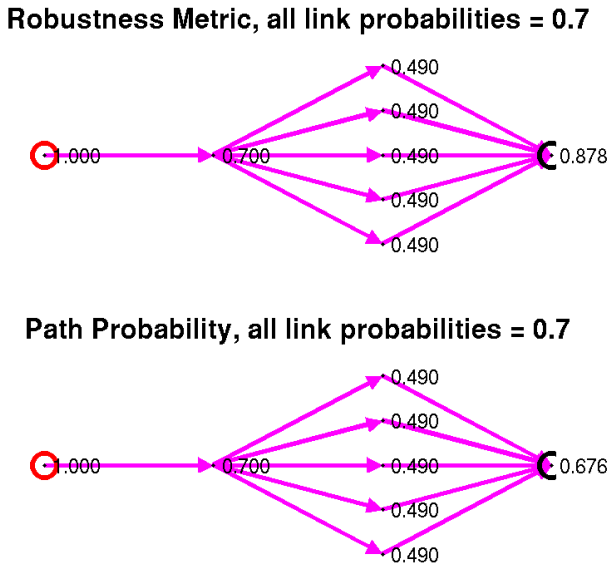

Figure 2.8. Serial-parallel topology with 5 parallel links, where the robustness metric at the sink $r_{a\to b}$ (top) is significantly greater than the path probability at the sink $p_{a\to b}$ (bottom). The source is circled in red, the sink is circled in black, and each node is labeled with $r_{a\to v}$ or $p_{a\to v}$ in the respective plot.

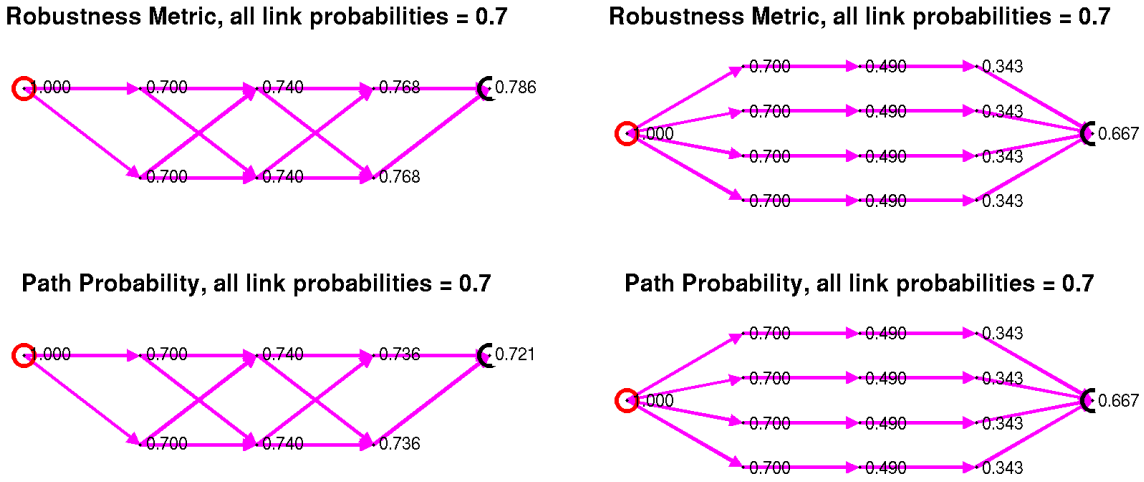

Figure 2.9. Comparison of a mesh topology (left) and a topology with disjoint paths (right) using the robustness metric $r_{a\to b}$ (top) and the path probability $p_{a\to b}$ (bottom). The source is circled in red, the sink is circled in black, and each node is labeled with $r_{a\to v}$ or $p_{a\to v}$ in the respective plot.

nodes in adjacent columns (typically a node has links to several nodes in the adjacent columns). We say the topology is a *width-k path topology* if there are k nodes in each column except the columns containing the source and the sink.

A comparison of $p_{a\to b}$ from Figures 2.10 and 2.11 shows that adding nodes to a topology

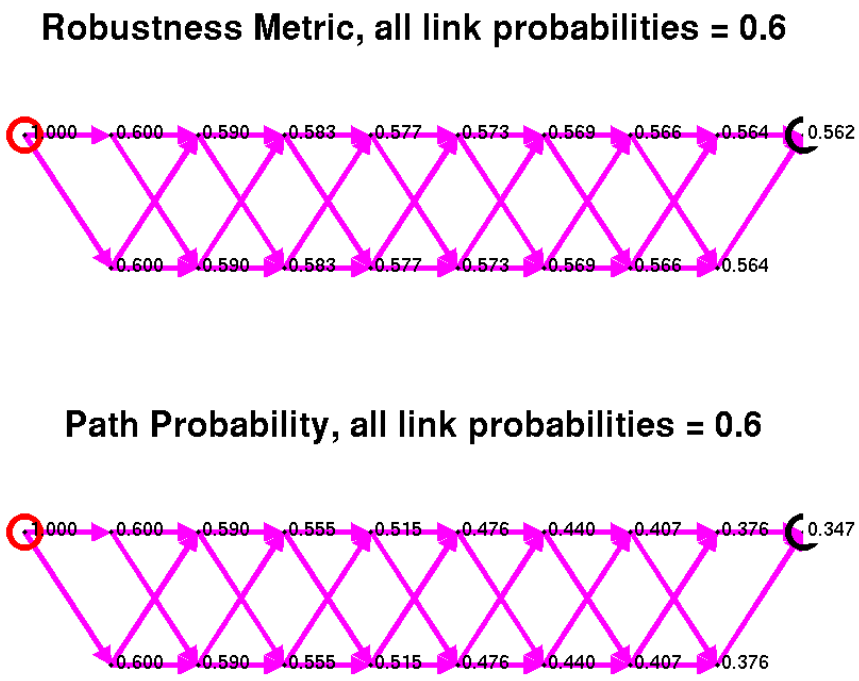

Figure 2.10. Width-2 path topology, every node in a column has an outgoing link to every node in the column to the right. The robustness metric at the sink $r_{a\to b}$ (top) is greater than the path probability at the sink $p_{a\to b}$ (bottom). The source is circled in red, the sink is circled in black, and each node is labeled with $r_{a\to v}$ or $p_{a\to v}$ in the respective plot.

to increase the number of interleaved paths is a good strategy for increasing the probability that a packet flooded from the source reaches the sink. The links in both topologies have an unusually low probability of $p_l = 0.6$ to show the dramatic difference between the topologies and to show that a significant gap can exist between $p_{a\to b}$ and $r_{a\to b}$ on each topology. The difference between $p_{a\to b}$ and $r_{a\to b}$ manifests itself more clearly in nodes that are far away from the source.

Also, notice in Figure 2.11 that the robustness metric of the nodes in each column appears to increase with the number of hops away from the source, while the path probability increases and then decreases. In fact, (2.5) on our width-3 path topology where all links have probability p_l simplifies to

$$\begin{aligned} r_{a\to a} &= 1 \\ r_{a\to v} &= 1 - (1 - p_l r_{a\to v'})^3 \end{aligned} \tag{2.7}$$

where v' is an upstream neighbor of v. It turns out there are two fixed points if we think

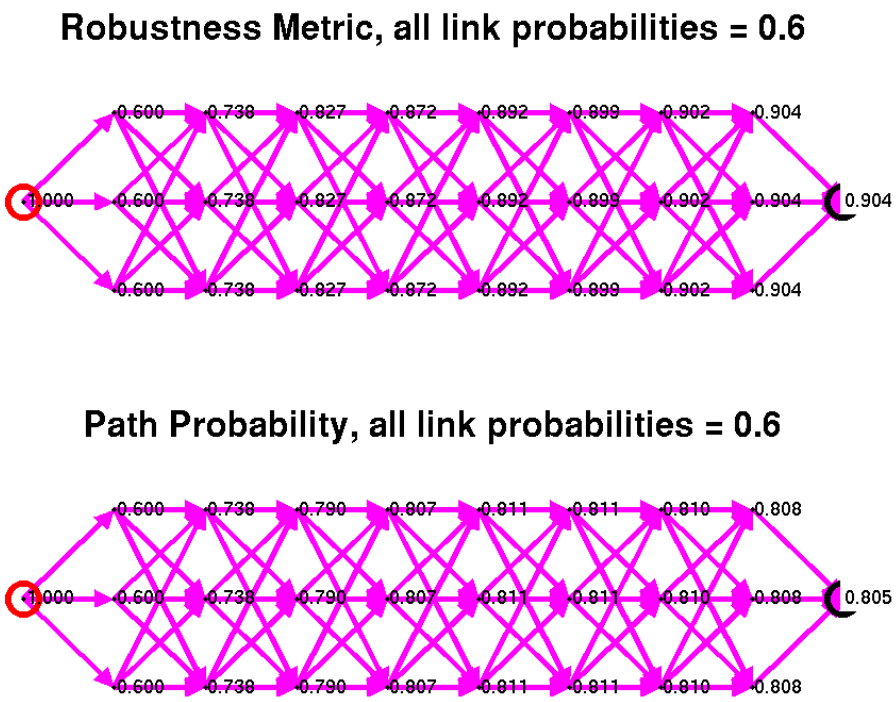

Figure 2.11. Width-3 path topology, every node in a column has an outgoing link to every node in the column to the right. The robustness metric at the sink $r_{a\to b}$ (top) is greater than the path probability at the sink $p_{a\to b}$ (bottom). The source is circled in red, the sink is circled in black, and each node is labeled with $r_{a\to v}$ or $p_{a\to v}$ in the respective plot.

of (2.7) as a continuous mapping from $[0,1]$ to $[0,1]$. These are 0 and 0.9043, the roots to the equation $1-(1-p_l x)^3 - x$ with variable x (the third root, 4.0957 is not in $[0,1]$). Clearly, $r_{a\to v}$ is approaching the fixed point 0.9043 as we look at nodes v farther from the source. On the other hand, the path probability will decrease toward 0 as we look at nodes farther away from the source, which is expected. The initial rise in path probability in the first five columns is a "transient effect" from having a single source which can flood the packet on three independent links (the path probability would start out higher and slowly decrease if we had three sources with the packet sending to all the nodes in the first column). In Chapter 4, after the Directed Staged Flooding model is presented and analyzed in Section 4.3.4, the reader will gain a better intuition of why this behavior is expected. In short, the robustness metric does not qualitatively match the characteristics of the path probability in some routing topologies. □

An interesting question is whether routing topologies that conform to a simple set of

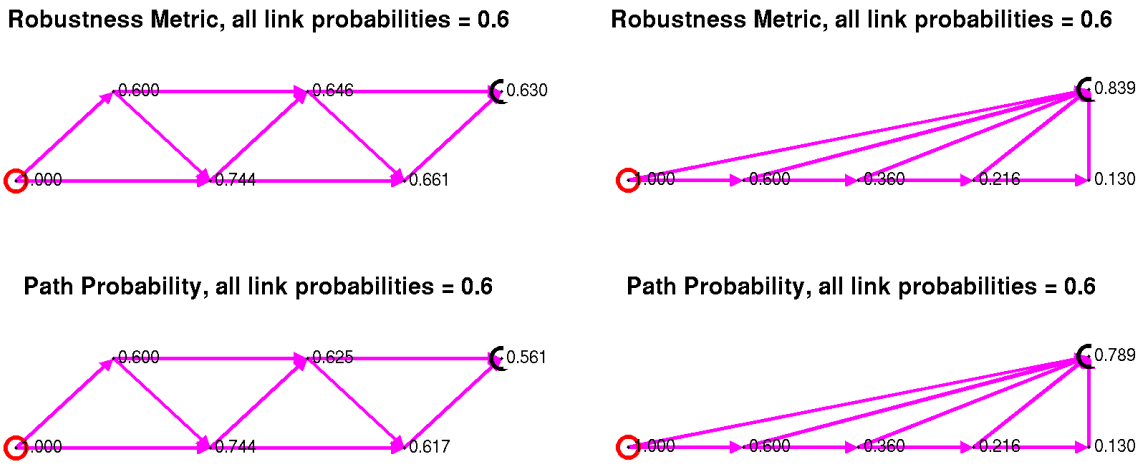

Figure 2.12. Two examples of two-parents-per-node topologies, where every node has two distinct downstream neighbors. The source is circled in red, the sink is circled in black, and each node is labeled with $r_{a\to v}$ or $p_{a\to v}$ in the respective plot.

constraints will have a high path probability from the source to the sink. For instance, the routing topology formation algorithm may reject any links with probability lower than a specified threshold and only allow nodes to join the network if it has links to at least two downstream nodes that have already joined the network. If we know the number of nodes in the network, can we lower bound the path probability from the source to the sink? What are some extreme examples of topologies under these constraints?

Example 2.2.4 (Topologies with downstream links constraint). Figure 2.12 depicts two topologies where each node, except for the sink and one node adjacent to the sink, have two links to distinct downstream neighbors. All links have probability 0.6, which represents the "worst case" when the threshold for a link to be used in the network is 0.6. Note that the topology on the right of Figure 2.12 depicts several nodes in a line for clarity, but you can think of them as surrounding the sink. If we were to add nodes to the network one by one, trying to maximize each node's minimum hop count path to the sink, we would get the topology on the left of Figure 2.12. If we were to add nodes to the network one by one, trying to connect to nodes that have the best path probability (or robustness metric) to the sink, we would get the topology on the right of Figure 2.12. The two topologies have very different path probabilities $p_{a\to b}$.

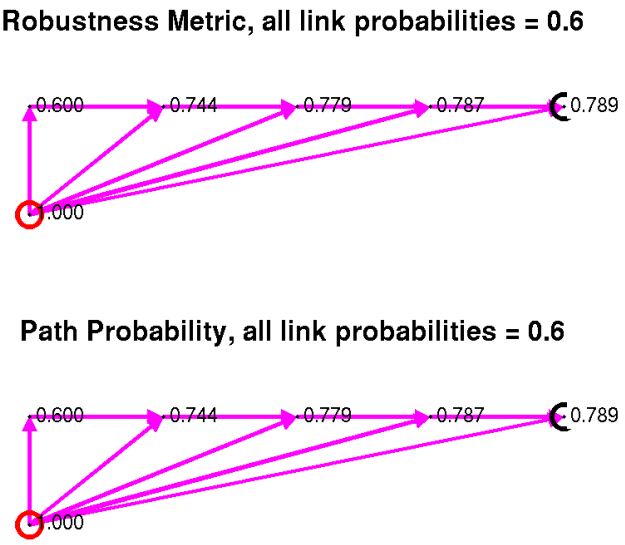

Figure 2.13. Two-children-per-node topology obtained by reversing the links on the topology on the right of Figure 2.12, where every node has two distinct upstream neighbors. The source is circled in red, the sink is circled in black, and each node is labeled with $r_{a\to v}$ or $p_{a\to v}$ in the respective plot.

Finally, to illustrate that $r^{G}_{a\to b} \neq r^{\overleftarrow{G}}_{b\to a}$, Figure 2.13 depicts a topology where every node, except the source and one node adjacent to the source, have two links to distinct *upstream* neighbors. This topology $\overleftarrow{G}$ is obtained by reversing the links of the topology G on the right of Figure 2.12. Although $r^{G}_{a\to b} \neq r^{\overleftarrow{G}}_{b\to a}$, notice that the path probability of the two topologies are equal.

□

In Section 2.4.3, we will compare the robustness metric and path probability on randomly generated graphs with routing topologies formed by an algorithm based on the robustness metric. Also, in Chapter 4 we will discuss how to compute an end-to-end connectivity metric for two types of TDMA routing protocols, Directed Staged Flooding (DSF) and Unicast Path Diversity (UPD). The connectivity metric for DSF closely resembles the computation of the path probability in Algorithm 2. In fact, if we set up a DSF schedule such that nodes transmit packets in the same order as they are selected by Algorithm 2, the connectivity metric after the last node has transmitted will match the path probability of the network. In this sense, the path probability of the network matches a communication

scheme where a node listens to all its parents (closer to the source) for a copy of the packet and broadcasts copies of the packet to all its children using a single transmission.

2.3 Unicast Flow Metrics

This section presents two *unicast flow metrics* for routing topologies, the *flow metric* and the *retransmission flow metric* (rtFlow metric, for short). Unlike the flooding connectivity metrics, these metrics assume unicast routing where only one copy of the packet travels down one path through the network, making decisions on which link to traverse at each node. This section will focus on the rtFlow metric, because the link failure model for the flow metric is less realistic. The flow metric is presented mainly as a motivation for the rtFlow metric.[7]

Under the link failure model for the rtFlow metric, it is possible that the packet reaches a node where all its outgoing links fail, i.e., the packet is "trapped at a node." Thus, topologies where a node is likely to receive a packet but has outgoing links with very low probabilities of success tend to perform poorly under unicast routing (See Example 2.3.2). Flooding is less affected by the phenomenon of "trapped packets" because other copies of the packets can still propagate down other paths. The rtFlow metric will thus "penalize" topologies that have a tendency to trap packets more heavily than the path probability metric would penalize these topologies. Like the path probability metric, the rtFlow metric shows that multiple interleaved paths typically provide better end-to-end connectivity than disjoint paths.

2.3.1 Routing and Link Failure Models

This section presents routing and link failure models for the flow metric and slightly different routing and link failure models for the retransmission flow metric. As in Section 2.2.1, in both link failure models all the links in the network fail independently and

[7] In fact, it is so simple that it probably has been developed by other researchers before. I didn't check the literature for this.

each link l succeeds with probability p_l. Also, in both routing models only one copy of the packet is routed through the network. The packets are routed following the links of the DAG describing the routing topology. Packets are routed without prior knowledge of which links have failed on the network, and cannot "backtrack" from a downstream node to an upstream node to find an alternate path.

In the flow metric link failure model, a link l succeeds or fails during a transmission time slot independently of whether it succeeded or failed during any previous time slot. In other words, at each time slot link l has no memory of its past, so the success or failure of the link over a sequence of time slots is a Bernoulli process with probability p_l of success.

The flow metric routing model assumes that when a packet reaches a node v, the node will select an outgoing link uniformly at random from all its outgoing links to transmit the packet. If the link fails, node v will again select an outgoing link uniformly at random from all its outgoing links, *including* the link that just failed (in other words, links are selected "with replacement" over time). Node v will continue to try to transmit on an outgoing link until it succeeds, which will eventually occur because an individual link's success or failure is independent of it's past successes or failures. Since packets cannot be lost, all packets transmitted from the source eventually reach the sink(s). The flow metric described in Section 2.3.2 is a crude measure of traffic distribution across the network under the described routing and link failure models.

In the retransmission flow metric link failure model, a link either succeeds or fails across *all* the time slots when the packet is being routed through the network. One interpretation is that the packet is routed through a particular realization of the collection of links in the network, where each link l is either up (success) or down (fail). The realization of the collection of links in the network for each packet is generated once by making each link l succeed with probability p_l.

The retransmission flow metric routing model assumes that after a packet reaches node v, the node will select an outgoing link uniformly at random from all its outgoing links to transmit the packet. If this link fails, node v will select another outgoing link uniformly at

random from all the outgoing links that *have not been selected for transmission before* (in other words, links are selected "without replacement" over time).[8] Each time an outgoing link fails, node v will select another outgoing link that has not been selected for transmission until either a transmission succeeds or all the outgoing links have been selected and all have failed. In the latter case, the packet is dropped from the network. Note that in this model each link is selected once for transmission and all permutations in the order of selecting links are equally probable. The retransmission flow metric described in Section 2.3.3 measures the proportion of packets that reach the nodes in the network under the described routing and link failure models.

2.3.2 Flow Metric

The flow metric $f_{a\to v} \in [0,1]$ measures the proportion of packets from node a that traverse through node v assuming the flow metric routing and link failure models. This can be computed given the *flow weights* on the links in the network, where w_{uv} is the flow weight for link (u,v). The flow weight of a link $l=(u,v)$ is the probability that a packet which arrived at node u traverses the link. The flow metric $f_{a\to v}$ is computed using the flow weights of node v's incoming links and the flow metrics of its upstream neighbors.

To compute w_{uv} where $l=(u,v)$, first let $\mathcal{E}_u = \{(u,v) \in \mathcal{E} : v \in \mathcal{V}\}$ be the set of outgoing links of node u and $\delta^+(u) = |\mathcal{E}_u|$ denote the number of outgoing links from node u. Let $A_e^{(k)}$ denote the event that node u selected link $e \in \mathcal{E}_u$ for the k-th attempt at transmitting the packet, and link e succeeds. Similarly, let $\bar{A}_e^{(k)}$ denote the event that node u selected link $e \in \mathcal{E}_u$ for the k-th attempt at transmitting the packet, but link e fails. Let $B^{(k)}$ denote the event that the k-th transmission failed.

Because the probability of selecting an outgoing link is uniform and independent of link success or failure, $\mathbb{P}(A_e^{(k)}) = \frac{1}{\delta^+(u)} p_e$ and $\mathbb{P}(\bar{A}_e^{(k)}) = \frac{1}{\delta^+(u)}(1-p_e)$. Since the events

[8] Given our link failure model, there is actually no difference if we select links with replacement or without replacement. If node v tries to transmit on a link l that was previously selected for transmission, it will fail again. This is as if link l was not selected at all. It is more intuitive to view this as selecting links without replacement.

$A_e^{(k)}, \forall e \in \mathcal{E}_u$ are disjoint,

$$\mathbb{P}(B^{(k)}) = \mathbb{P}\left(\bigcup_{e\in\mathcal{E}_u} \bar{A}_e^{(k)}\right) = \sum_{e\in\mathcal{E}_u} \mathbb{P}(\bar{A}_e^{(k)}) = \sum_{e\in\mathcal{E}_u} \frac{1-p_e}{\delta^+(u)} \quad .$$

Notice that the events $B^{(k)}, k = 1, 2, \ldots$ all have equal probability, and the events $A_l^{(k)}, k = 1, 2, \ldots$ for a fixed l all have equal probability.

The probability that a packet traverses down link l is thus

$$\begin{aligned} w_{uv} &= \mathbb{P}\left(A_l^{(1)} \cup (B^{(1)} \cap A_l^{(2)}) \cup (B^{(1)} \cap B^{(2)} \cap A_l^{(3)}) \cup \cdots\right) \\ &\overset{(a)}{=} \mathbb{P}(A_l^{(1)}) + \mathbb{P}(B^{(1)} \cap A_l^{(2)}) + \mathbb{P}(B^{(1)} \cap B^{(2)} \cap A_l^{(3)}) + \cdots \\ &\overset{(b)}{=} \mathbb{P}(A_l^{(1)}) + \mathbb{P}(B^{(1)}) \cdot \mathbb{P}(A_l^{(2)}) + \mathbb{P}(B^{(1)}) \cdot \mathbb{P}(B^{(2)}) \cdot \mathbb{P}(A_l^{(3)}) + \cdots \end{aligned}$$

where step (a) is because the corresponding events are disjoint and step (b) is because the corresponding events are independent. Substituting in probabilities, we get

$$w_l = \frac{p_l}{\delta^+(u)} + \left(\frac{\sum_{e\in\mathcal{E}_u}(1-p_e)}{\delta^+(u)}\right)\frac{p_l}{\delta^+(u)} + \left(\frac{\sum_{e\in\mathcal{E}_u}(1-p_e)}{\delta^+(u)}\right)^2\frac{p_l}{\delta^+(u)} + \cdots$$

which is a geometric series. Recall that if $\alpha < 1$ then $1 + \alpha + \alpha^2 + \cdots = \frac{1}{1-\alpha}$, so

$$w_l = \frac{p_l}{\delta^+(u)}\left(\frac{1}{1 - \frac{\sum_{e\in\mathcal{E}_u}(1-p_e)}{\delta^+(u)}}\right) = \frac{p_l}{\sum_{e\in\mathcal{E}_u} p_e} \quad .$$

Definition 2.3.1 (Flow Metric)**.** Let $G = (\mathcal{V}, \mathcal{E}, p)$ be a weighted DAG where all nodes have an outgoing edge except for the destination node b, and let p_{ij} be the probability of link (i, j). The *flow metric* $f_{a\to b} \in [0, 1]$ on G with respect to a source-destination pair (a, b) can be computed using

$$\begin{aligned} f_{a\to a} &= 1 \\ f_{a\to v} &= \sum_{u_k} f_{a\to u_k} w_{u_k v} \end{aligned} \tag{2.8}$$

where u_k are all the upstream neighbors of node v and $w_{u_k v}$ is the *flow weight* of link $l = (u_k, v)$. The flow weight of link $l = (u, v)$ is given by

$$w_{uv} = \frac{p_l}{\sum_{e\in\mathcal{E}_u} p_e} \tag{2.9}$$

where $\mathcal{E}_u = \{(u, v) \in \mathcal{E} : v \in \mathcal{V}\}$ denotes the set of outgoing links of node u. When there are several graphs, we use the notation $f_{a\to v}^G$ to specify $f_{a\to v}$ on graph G. □

It is easy to see from (2.8) and (2.9) that $f_{a\to b} = 1$ when b is the only sink in the routing topology DAG. Therefore, $f_{a\to b}$ is not a good metric to compare two different routing topologies to see which is more likely to successfully deliver packets to the sink. One might be tempted to incorporate packet loss into the flow metric by multiplying the flow weights w_l by the link probability p_l and using that as a new flow weight. However, this approach to incorporate packet loss into the flow metric does not distinguish between a single link topology from a parallel link topology (Figure 2.5, right) when all links have probability p.

2.3.3 Retransmission Flow Metric

The retransmission flow metric (rtFlow metric) $\varrho_{a\to v} \in [0, 1]$ measures the proportion of packets from node a that traverse through node v assuming the retransmission flow metric routing and link failure models. Similar to the flow weight, the retransmission flow weight (rtFlow weight) for link $l = (u, v)$ is the probability that a packet which arrived at node u traverses link l, and is denoted $\varpi_l \in [0, 1]$. $\varrho_{a\to v}$ is computed using the rtFlow weights of node v's incoming links and the rtFlow metrics of its upstream neighbors.

For more compact notation in our following derivation of the rtFlow metric from the routing and link failure models, we will use random variables as "indicator functions" of events. For example, the random variable $\boldsymbol{X}$ will take on the value 1 when the event associated with $\boldsymbol{X}$ occurs and 0 otherwise. We will use the compact notation $\mathbb{P}_{\boldsymbol{X},\boldsymbol{Y}}(x, y)$ to denote $P(\boldsymbol{X} = x, \boldsymbol{Y} = y)$, where $\boldsymbol{X}, \boldsymbol{Y}$ are random variables. Because we are using the random variables as indicator functions, we will sometimes abuse the notation and terminology by referring to the value taken by a random variable or a set of random variables (e.g. x in the previous expression) as an event.

Let the random variable $\boldsymbol{A}_l$ equal 1 when link l is the first successful link in the sequence of link transmissions chosen by node u and 0 otherwise. The event associated with $\boldsymbol{A}_l$ is the same as the event that a packet at node u traverses link l, so $\varpi_l = \mathbb{P}_{\boldsymbol{A}_l}(1)$. To compute ϖ_l, first let L denote the number of outgoing links from node u ($L = \delta^+(u)$, the outdegree

of node u) and with abuse of notation let's refer to u's outgoing links as $1, \ldots, L$ (in the following discussion, we will also let link l be a number, $l \in \{1, \ldots, L\}$ instead of $l = (u, v)$). The order which node u will try to transmit on its outgoing links can be represented by a permutation of the links $1, \ldots, L$. Let the random variable $\boldsymbol{P}$ map each permutation of links $1, \ldots, L$ to a number in $\{1, \ldots, L!\}$ (the actual mapping does not matter in the following discussion). Let the random variable $\boldsymbol{B}_i$ equal 1 when link i succeeds and 0 when it fails. Recall that links do not change from "success" to "fail" or vice versa during the time a packet is routed through the network. $\boldsymbol{A}_l$ is a function of the random variables $\boldsymbol{P}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_L$. The rtFlow weight is

$$
\begin{aligned}
\varpi_l &= \mathbb{P}_{\boldsymbol{A}_l}(1) \\
&= \sum_{\pi, b_1, \ldots, b_L} \mathbb{P}_{\boldsymbol{A}_l, \boldsymbol{P}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_L}(1, \pi, b_1, \ldots, b_L) \\
&= \sum_{b_1, \ldots, b_L} \sum_{\pi} \mathbb{P}_{\boldsymbol{A}_l | \boldsymbol{P}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_L}(1 | \pi, b_1, \ldots, b_L) \cdot \\
&\quad \mathbb{P}_{\boldsymbol{P} | \boldsymbol{B}_1, \ldots, \boldsymbol{B}_L}(\pi | b_1, \ldots, b_L) \cdot \mathbb{P}_{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_L}(b_1, \ldots, b_L) \\
&\overset{(c)}{=} \sum_{b_1, \ldots, b_L} \sum_{\pi} \mathbb{P}_{\boldsymbol{A}_l | \boldsymbol{P}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_L}(1 | \pi, b_1, \ldots, b_L) \cdot \mathbb{P}_{\boldsymbol{P}}(\pi) \cdot \\
&\quad \mathbb{P}_{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_L}(b_1, \ldots, b_L) \\
&= \sum_{b_1, \ldots, b_L} \Big(\underbrace{\sum_{\pi} \overbrace{\mathbb{P}_{\boldsymbol{A}_l | \boldsymbol{P}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_L}(1 | \pi, b_1, \ldots, b_L)}^{(d)} \cdot \mathbb{P}_{\boldsymbol{P}}(\pi)}_{(e)} \Big) \cdot \\
&\quad \mathbb{P}_{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_L}(b_1, \ldots, b_L)
\end{aligned}
$$

where step (c) is because the order of transmission attempts on node u's outgoing links is independent of the collection of link successes and failures, i.e., $\boldsymbol{P} \perp\!\!\!\perp (\boldsymbol{B}_1, \ldots, \boldsymbol{B}_L)$. The summations over $b_1, \ldots, b_L$ are for $b_i \in \{0, 1\}, \forall i$ and similarly the summation over π is for $\pi \in \{1, \ldots, L!\}$.

The next step is to simplify expression (e). First, let $\mathcal{E}_u^* = \{e_1, \ldots, e_E\}$ denote the set of successful links from the event $(b_1, \ldots, b_L)$, where $E = |\mathcal{E}_u^*|$. Expression (d) serves as an indicator function of whether $(\pi, b_1, \ldots, b_L)$ is an event where link l is the first successful link in the permutation π. Therefore, expression (e) is a weighted sum that counts the number

of permutations where link l is ordered before any other successful link from $\mathcal{E}_u^*$, and each count is weighted by the probability of the permutation. Note that for all $e, e' \in \mathcal{E}_u^*$, the number of permutations where link e is ordered before all the links in $\mathcal{E}_u^* \backslash e$ is equal to the number of permutations where link e' is ordered before all the links in $\mathcal{E}_u^* \backslash e'$. Also, all permutations π have equal probability. Finally, if we let A_{e_j} denote the event that includes all permutations π where link e_j is ordered before all the links in $\mathcal{E}_u^* \backslash e_j$, the events $A_{e_1}, \ldots, A_{e_E}$ partition the sample space of all permutation events. This leads us to conclude that $\mathbb{P}(A_{e_j}) = \frac{1}{E}, \forall e_j \in \mathcal{E}_u^*$. Using $\boldsymbol{B_l}$ as an indicator function of when link l is successful, expression (e) simplifies to $\boldsymbol{B_l} \cdot \mathbb{P}(A_{e_j}) = \frac{\boldsymbol{B_l}}{E}$. Since $\sum_{k=1}^{L} b_k = E$,

$$\varpi_l = \sum_{b_1,\ldots,b_L} \frac{\boldsymbol{B_l}}{\sum_{k=1}^{L} b_k} \cdot \mathbb{P}_{\boldsymbol{B}_1,\ldots,\boldsymbol{B}_L}(b_1, \ldots, b_L) \qquad .$$

Recall that the links succeed or fail independently, so

$$\mathbb{P}_{\boldsymbol{B}_1,\ldots,\boldsymbol{B}_L}(b_1, \ldots, b_L) = \mathbb{P}_{\boldsymbol{B}_1}(b_1) \cdots \mathbb{P}_{\boldsymbol{B}_L}(b_L)$$

where

$$\mathbb{P}_{\boldsymbol{B}_i}(b_i) = \begin{cases} p_i & \text{if } b_i = 1 \\ 1 - p_i & \text{if } b_i = 0 \end{cases} \qquad .$$

We get the following definition of the retransmission flow metric after a change of variables and expressions used in the derivation (e.g., $\mathcal{E}' = \mathcal{E}_u^* \backslash l$).

Definition 2.3.2 (Retransmission Flow Metric)**.** Let $G = (\mathcal{V}, \mathcal{E}, p)$ be a weighted DAG where all nodes have an outgoing edge except for the destination node b, and let p_{ij} be the probability of link (i, j). The *retransmission flow metric* (rtFlow metric) $\varrho_{a \to b} \in [0, 1]$ on G with respect to a source-destination pair (a, b) can be computed using

$$\begin{aligned} \varrho_{a \to a} &= 1 \\ \varrho_{a \to v} &= \sum_{u_k} \varrho_{a \to u_k} \varpi_{u_k v} \end{aligned} \tag{2.10}$$

where u_k are all the upstream neighbors of node v and $\varpi_{u_k v} \in [0, 1]$ is the *retransmission flow weight* (rtFlow weight) of link (u_k, v). The rtFlow weight for link $l = (u, v)$ is given by

$$\varpi_{uv} = \sum_{\mathcal{E}' \in 2^{\mathcal{E}_u \backslash l}} \frac{p_l}{|\mathcal{E}'| + 1} \left(\prod_{e \in \mathcal{E}'} p_e \right) \left(\prod_{\bar{e} \in \mathcal{E}_u \backslash (\mathcal{E}' \cup l)} 1 - p_{\bar{e}} \right) \qquad . \tag{2.11}$$

Here, $\mathcal{E}_u = \{(u,v) \in \mathcal{E} : v \in \mathcal{V}\}$, all the outgoing links of node u.

If the links $(u,v) \in \mathcal{E}_u$ all have probability p, then (2.11) simplifies to

$$\varpi_{uv} = \sum_{k=1}^{\delta^+(u)} \frac{1}{k} \binom{\delta^+(u)-1}{k-1} p^k (1-p)^{\delta^+(u)-k} \tag{2.12}$$

where $\delta^+(u)$ is the outgoing degree of node u.

When there are several graphs, we use the notation $\varrho^G_{a\to v}$ to specify $\varrho_{a\to v}$ on graph G.

The *link retransmission flow metric* (link rtFlow metric) $\varrho_{uv} \in [0,1]$ of a link (u,v) gives the amount of flow down a link and is

$$\varrho_{uv} = \varrho_{a\to u} \varpi_{uv} \qquad . \tag{2.13}$$

□

2.3.4 Examples and Discussion

The following example shows how to compute flow weights and rtFlow weights and gives some insights and strategies for modifying the routing topology to improve the rtFlow metric.

Example 2.3.1 (Unicast flow metrics on width-3 path topologies). Figure 2.14 shows a graph topology where every node but the sink and its upstream neighbors has three outgoing links. Using (2.9), the flow weights of a node that has outgoing links $1,2,3$ with link probabilities p_1, p_2, p_3 respectively would be

$$\begin{aligned} w_1 &= \frac{p_1}{p_1+p_2+p_3} \\ w_2 &= \frac{p_2}{p_1+p_2+p_3} \\ w_3 &= \frac{p_3}{p_1+p_2+p_3} \qquad . \end{aligned} \tag{2.14}$$

Similarly, using (2.11) the rtFlow flow weights are

$$\begin{aligned} \varpi_1 &= \frac{1}{3}p_1p_2p_3 + \frac{1}{2}(p_1p_2\bar{p}_3 + p_1\bar{p}_2p_3) + p_1\bar{p}_2\bar{p}_3 \\ \varpi_2 &= \frac{1}{3}p_1p_2p_3 + \frac{1}{2}(p_1p_2\bar{p}_3 + \bar{p}_1p_2p_3) + \bar{p}_1p_2\bar{p}_3 \\ \varpi_3 &= \frac{1}{3}p_1p_2p_3 + \frac{1}{2}(\bar{p}_1p_2p_3 + p_1\bar{p}_2p_3) + \bar{p}_1\bar{p}_2p_3 \end{aligned} \tag{2.15}$$

where we use the shorthand $\bar{p}_i = 1 - p_i$. Note that the formulas for the flow and rtFlow metric can also be applied to networks with multiple frequency channels, offering multiple independent channel-links between nodes in the network (in our previous definition of a graph, at most one link can exist between a pair of nodes). Each channel-link is treated as an independent link, and the calculations for the flow weight and the flow metric proceed as before.

Figure 2.14 shows the flow metric and rtFlow metric for each node in a meshed paths topology. All links have the same probability in the topology on the left while the links have a range of probabilities in the topology on the right. Both flow and rtFlow metrics for the topology on the left shows an even packet distribution throughout the network since the network is symmetric and all the link probabilities are equal. Both metrics for the topology on the right also shows fairly even packet distribution, despite the variation in link probabilities.

A quick comparison of Figure 2.15 and the left of Figure 2.14, where both topologies have the same link probabilities and the same number of links between the intermediate columns, confirms that meshed topologies tend to have a higher probability of delivering a unicast packet under the rtFlow metric routing and link failure models than disjoint paths topologies.

Notice that in the topology on the left of Figure 2.14, the rtFlow metric at the sink is less than or equal to 0.9. In general, if the upstream neighbors of the sink do not have links between each other and they all have one link to the sink, the rtFlow metric at the sink cannot be greater than the largest probability of all its incoming links. This is because the sum of the rtFlow metric across all the upstream neighbors of the sink is less than or equal to 1. The rtFlow metric on a node v represents the probability that a packet visits v as it is routed through the network, and the packet can only visit one of the upstream neighbors of a sink in any path it traverses through the network (visiting the upstream neighbors are disjoint events).

These observations point to three strategies at the sink that can increase the rtFlow

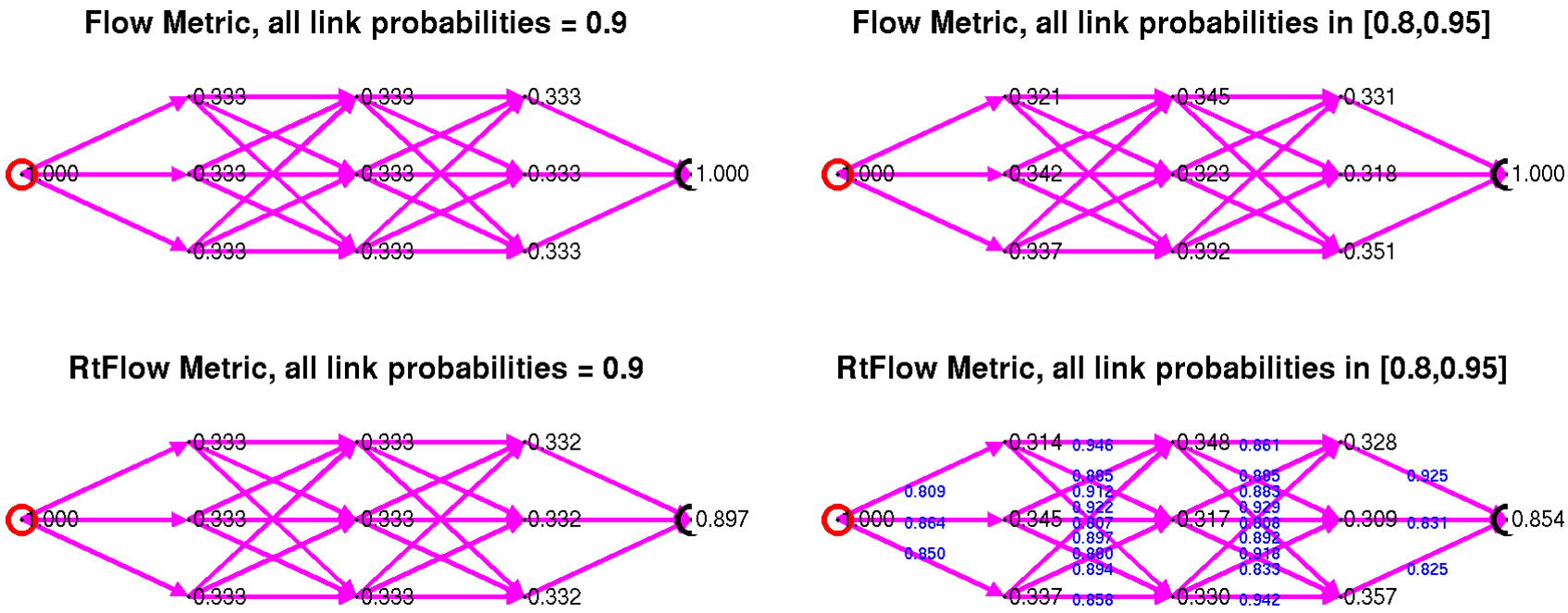

Figure 2.14. Comparison of two wide path routing topologies where (left) all links have probability $p_l = 0.9$ and where (right) all links have probability p_l selected uniformly at random from the range $[0.8, 0.95]$. The comparisons are done using the flow metric $f_{a\to b}$ (top) and the retransmission flow metric $\varrho_{a\to b}$ (bottom). The source is circled in red, the sink is circled in black, and each node is labeled in black with $f_{a\to v}$ or $\varrho_{a\to v}$ in the respective plot. The links of the topology on the right are labeled in blue (smaller font) with their link probabilities.

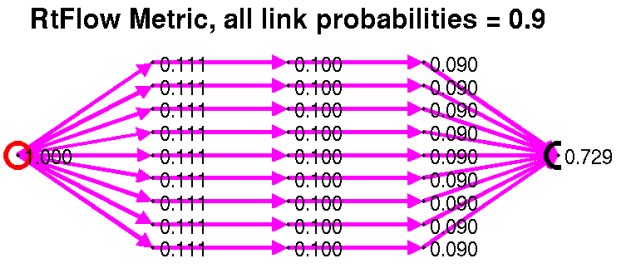

Figure 2.15. rtFlow metric on a parallel disjoint paths topology, for comparison with the topology on the left of Figure 2.14. The source is circled in red, the sink is circled in black, and nodes are labeled by $\varrho_{a\to v}$.

metric $\varrho_{a\to b}$. We can add links between the upstream neighbors of the sink such that a node can try to relay the packet to another upstream neighbor if it fails to transmit to the sink (Figure 2.16). Also, we can add multiple sinks to the network connected by a wired backbone (so they behave as one sink with two receivers at two different locations), such that each upstream neighbor of the sinks has two outgoing links, one to each sink (Figure 2.17). Otherwise, we can use multiple frequency channels to create multiple independent (or approximately independent) channel-links to the sink from each of its upstream neighbors.

□

Example 2.3.2 (Adding lossy paths). Figure 2.18 gives a topology where an increase in a link's probability actually *lowers* the rtFlow metric. This illustrates the phenomenon of "trapped packets" mentioned at the beginning of Section 2.3. In this example, an increase in the link probability boxed in green reflects that, on average, more packets will arrive at the intermediate node with a low probability outgoing link. When that outgoing link fails, it is more likely to trap a packet at the node. In the same fashion, sometimes adding a link to a topology can decrease $\varrho_{a\to b}$. This is different from the path probability and robustness metrics, where adding a link to a topology or increasing the probability of a link can never lower the metric between the source and the sink.

□

The rtFlow metric is useful during the deployment stage to estimate the likelihood that

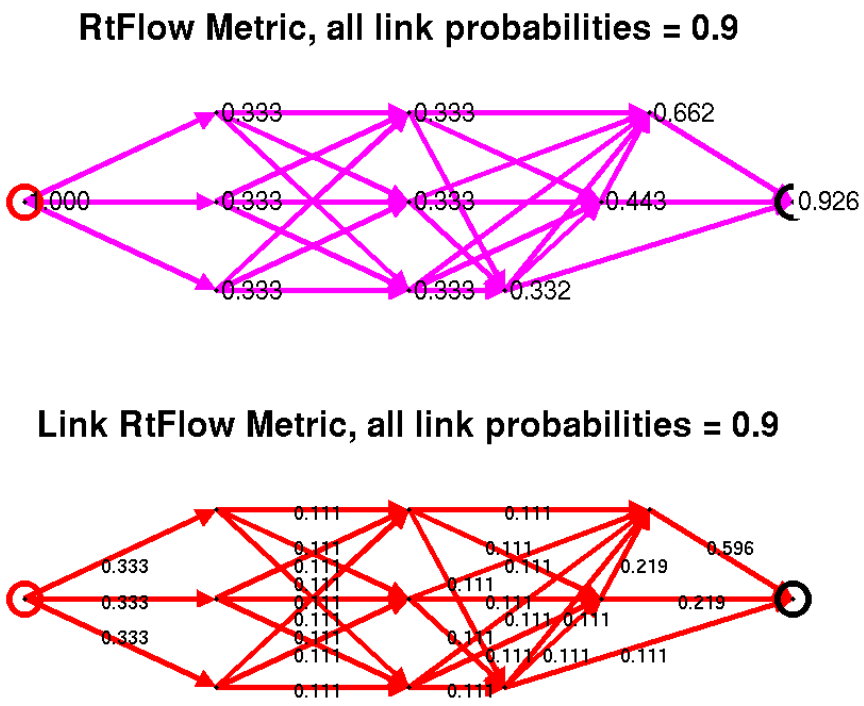

Figure 2.16. rtFlow metric on width-3 path topology with links added between the upstream neighbors of the sink, where all links have probability $p_l = 0.9$. .

a packet from the source can reach the sink. We consider again the routing topologies from Example 2.2.4 to compare the rtFlow metric and the path probability metric.

Example 2.3.3 (Topologies with downstream links constraint). Figure 2.19 shows the rtFlow metric on the topologies in Figure 2.12, and Figure 2.20 shows the rtFlow metric on the topology in Figure 2.13. Notice that $\varrho_{a\to b}$ is lower on the left topology of Figure 2.19 than the right topology of Figure 2.19, which is consistent with the path probability on the two topologies. Since the routing model of the rtFlow metric sends one packet through the network while the routing model of the path probability metric sends multiple copies of the packet through the network, given a topology G, $\varrho^G_{a\to b} \leq p^G_{a\to b}$. One interpretation is that the rtFlow metric penalizes trapped packets more heavily than the path probability.

Figure 2.20 and the right of Figure 2.19 together show that, like the robustness metric, there exist graphs G where $\varrho^G_{a\to b} \neq \varrho^{\overleftarrow{G}}_{b\to a}$.

□

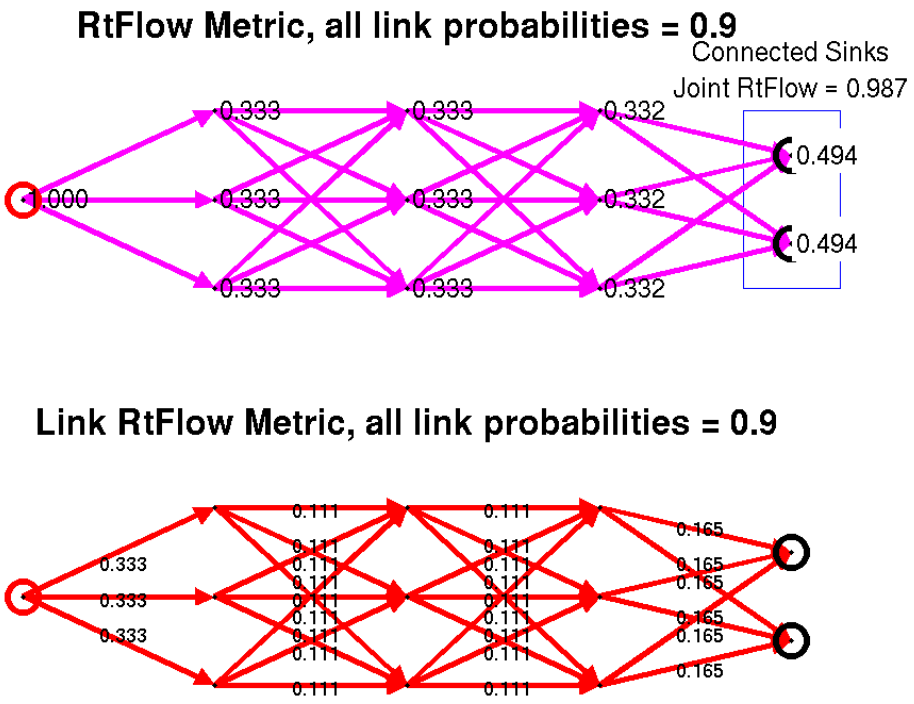

Figure 2.17. rtFlow metric on width-3 path topology with an additional sink, where all links have probability $p_l = 0.9$. The two sinks are connected by a wired backbone, so the rtFlow metric on the two sinks is the sum of rtFlow metric on each sink.

Unfortunately, the computation required to compute the rtFlow weights using (2.11) is exponential in the number of outgoing links of a node. This may not be a problem if the network restricts the number of outgoing links on each node in the network (e.g., ≤ 10 links). The following example examines a slight variation to the routing model that result in a more computationally tractable variant of the rtFlow metric.

Example 2.3.4 (Alternate flow weights). One alternate routing model would be for a node to always attempt transmission on outgoing links in decreasing order of link probabilities. As before, the node would try each link once and drop the packet when all links fail. This model leads to the following *probability-ordered rtFlow metric* (po-rtFlow), $\varrho'_{a\to v}$ calculated in the same fashion as the rtFlow metric except ϖ_{uv} is replaced by

$$\varpi'_{l_i} = \prod_{k=1}^{i-1}(1 - p_{l_k})p_{l_i} \tag{2.16}$$

where the outgoing links of node u have been sorted from highest to lowest probability into the list $(l_1, \ldots, l_{\delta^+(u)})$.

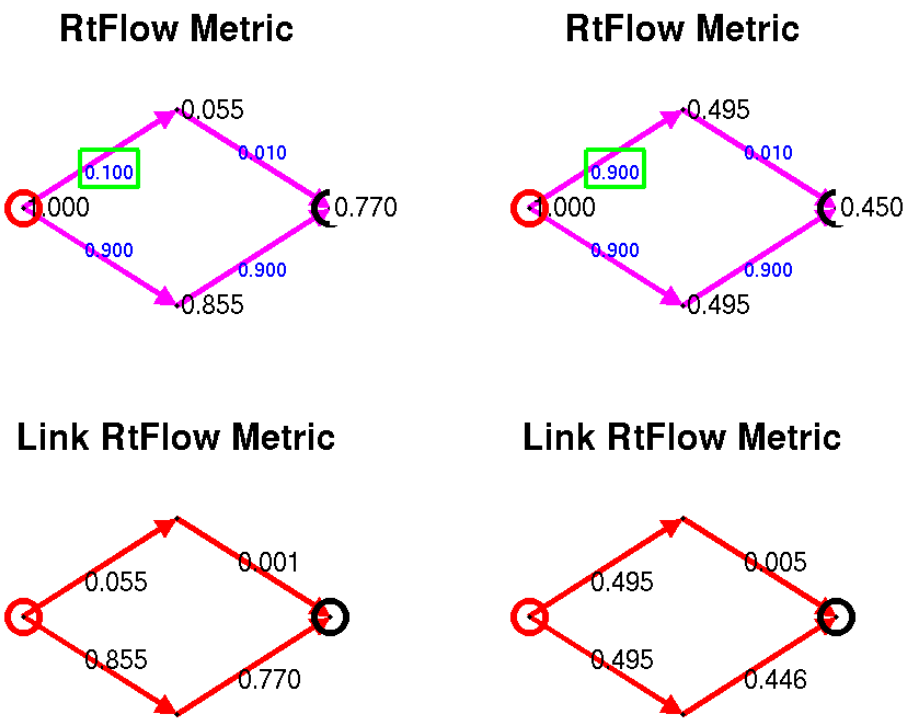

Figure 2.18. Example where increasing the probability of a link l (boxed in green) can decrease the rtFlow metric. The plots in the same column correspond to topologies with the same p_l. In the plots on the top, the nodes are labeled in black with the rtFlow metric and the links are labeled in blue (smaller font) with the link probability. In the plots on the bottom, the links are labeled with the link rtFlow metric.

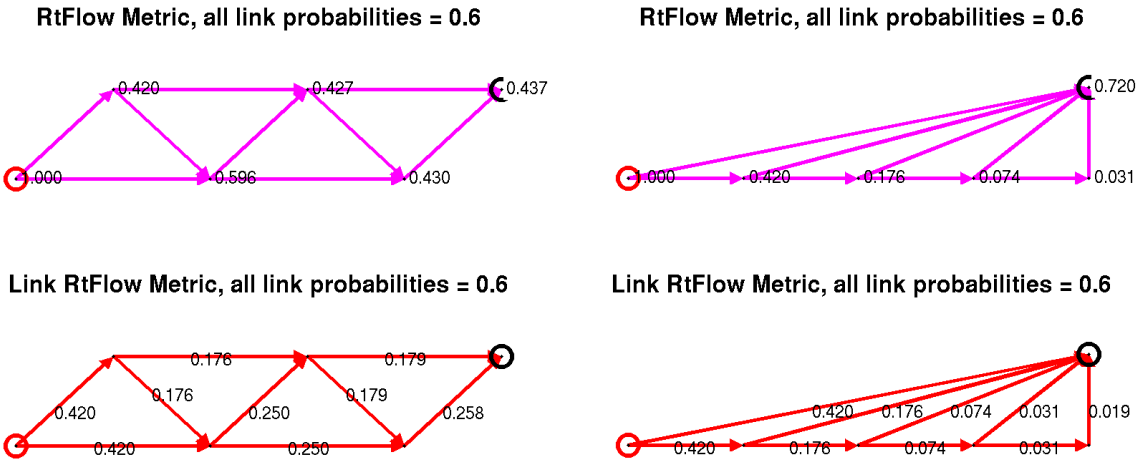

Figure 2.19. Two examples of two-parents-per-node topologies, where every node has two distinct downstream neighbors. The source is circled in red, the sink is circled in black, node labels on the top plots correspond to $\varrho_{a\to v}$, and link labels on the bottom plots correspond to ϱ_{uv}.

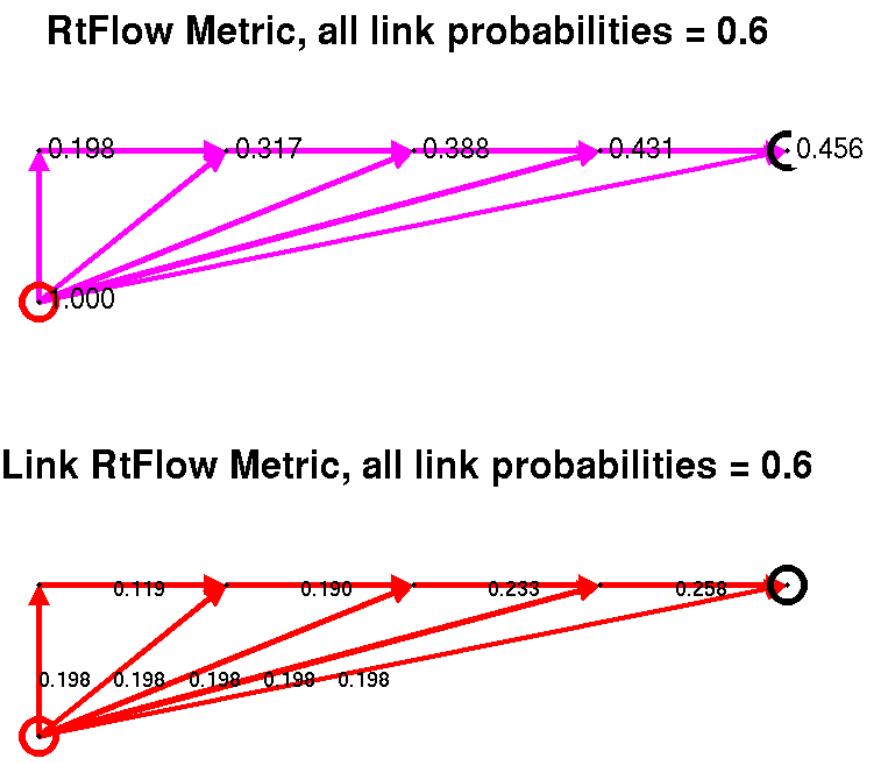

Figure 2.20. Two-children-per-node topology obtained by reversing the links on the topology on the right of Figure 2.19, where every node has two distinct upstream neighbors. The source is circled in red, the sink is circled in black, node labels on the top plot correspond to $\varrho_{a\to v}$, and link labels on the bottom plot correspond to ϱ_{uv}.

When we calculate $\varrho'_{a\to v}$ on the same topology (and same link probabilities) from the right of Figure 2.14 we get Figure 2.21. Notice that the traffic is far less evenly distributed on the nodes in the network. Also, in this example $\varrho'_{a\to v} < \varrho_{a\to v}$, but there are other examples where $\varrho'_{a\to v} > \varrho_{a\to v}$. In other words, from the point of view of the probability that a packet from the source reaches the destination, one routing model is not better than the other. However, because the traffic is less evenly distributed if there is a fixed order of link transmission attempts, we expect the probability-ordered routing model to be more sensitive to link estimation errors of outgoing links at nodes that carry most of the traffic.

□

One consideration when proposing routing models for calculating flow weights is whether it is likely to match the schedule generated by the scheduling algorithm after the routing

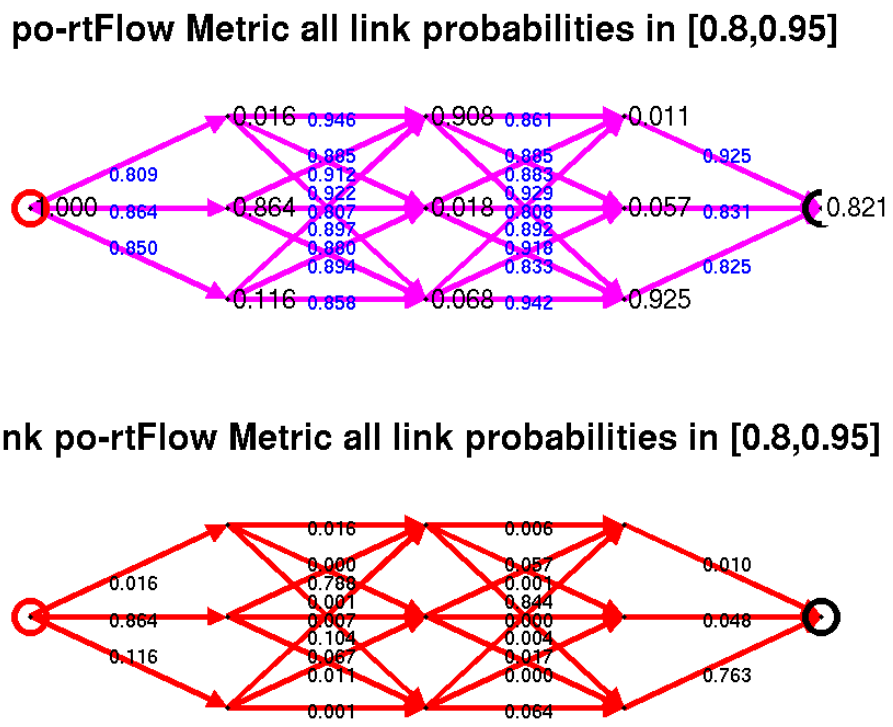

Figure 2.21. Probability-ordered rtFlow metric on a width-3 path topology where all links have probability p_l selected uniformly at random from the range $[0.8, 0.95]$. The source is circled in red, the sink is circled in black, node labels on the top plot correspond to $\varrho'_{a\to v}$, link labels on the top plot correspond to link probabilities, and link labels on the bottom plot correspond to ϱ'_{uv}.

topology has been formed. Typically, a node that has received a packet at time t will want to transmit the packet on the first outgoing link scheduled to transmit after time t, so as to reduce the end-to-end packet latency. As we will see in the next chapter, a fixed, repeating schedule will often route packets down a few paths in networks, but exactly which paths (and links) are used more frequently may be hard to predict while building the routing topology. Hence, it is hard to order the outgoing links for computing flow weights, as we tried to do in the previous example. On the other hand, the rtFlow metric routing model is meant to match randomized schedules where at any time t, the next scheduled outgoing link is selected uniformly from all the outgoing links.

2.4 Generating a Routing Topology

This section describes an algorithm that uses a variant of the robustness metric to construct a *robust routing topology* from an undirected connectivity graph. The robust routing topology is a DAG for routing packets from all the nodes in the network to the sink. The goal of the algorithm is to generate a routing topology such that packets from the nodes in the network are likely to reach the sink.

We assume the all the nodes in the connectivity graph are connected and all links are bi-directional, meaning that a receiver can send acknowledgments to the transmitter. Any uni-directional links will not be used by our routing algorithm. This is not a big limitation because studies have shown that high quality links are often bidirectional [14]. In fact, poor links may be pruned from the connectivity graph before the connectivity graph is presented to the routing topology construction algorithm. Also, we always assume the timescale at which the underlying undirected graph changes significantly (enough to warrant regenerating the routing topology) is much larger than the time it takes to deliver a packet through the network. Thus, the routing topology is assumed to be static.

In this section, we first review a simple routing topology, the *minimum hop DAG*, to motivate the *robust DAG* routing topology.

2.4.1 Minimum Hop Topology

An important criterion when generating a routing topology is to ensure there are no routing loops. The routing topology must therefore be a DAG. A commonly used method (not invented by the author) to generate a DAG is to first generate a routing tree, then label all the nodes by the number of hops away from the root, then orient all the links on the graph from nodes with a higher hop count to nodes with a lower hop count, and then orient the links between nodes with the same hop count such that no routing loops are formed. The initial routing tree can easily be generated by having the sink serve as the root node and broadcast a message with hop count 0, then having all nodes that hear that message

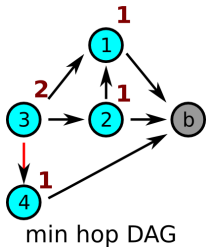

Figure 2.22. A minimum hop DAG routing topology. Nodes are labeled by their minimum hop count (not encircled). Node 4 would have an alternate path to the root if link $(3, 4)$ were reversed.

broadcast a message with hop count 1, and so forth. Let $\boldsymbol{h} = [h_1\ h_2\ \cdots\ h_N]$ be the vector of hop counts for nodes $1, \ldots, N$, where node i has hop count h_i.

To send a packet from a node to the root (the sink), we would route messages following the links of the graph, passing messages from nodes with higher hop counts to nodes with lower hop counts. We call the resulting routing topology a *minimum hop DAG*. Note that in such a topology, a node v with a hop count h_v can only send messages to nodes with a hop count of h_v or $h_v - 1$. That is, if v could send messages to a node u with a hop count $h_u < h_v - 1$, then v would have a hop count $h_u + 1$. Figure 2.22 shows that a minimum hop count DAG may not always provide enough paths for a robust connection to the root for all nodes. This motivates the next section, where we try to construct a *robust DAG*, or a *robust routing topology*.

2.4.2 Robust Topology

As seen above, assigning an ordering to the nodes in a network using hop counts and enforcing rules based on the hop counts, for instance a node must only route to another node with a lower hop count, is a useful technique to help ensure that there are no routing loops in the topology. In fact, we do not have to set a node's hop count equal to it's minimum hop count. The minimum hop count is interesting, however, because the nodes with the same minimum hop count h form a vertex cut of the network separating the source from the sink (if the source a has a minimum hop count $h_a > h$). This means that given the connectivity graph for the network, for all $h \in \{1, \ldots, h_a - 1\}$ the packet must visit at least

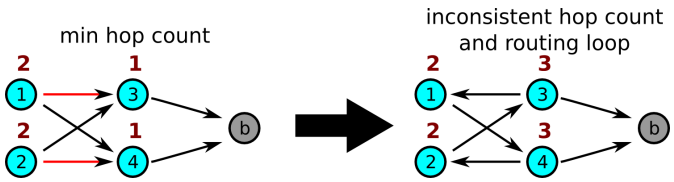

Figure 2.23. Nodes are labeled by their hop count. In this scenario, node 3 and 4 both decide simultaneously that they need to demote their hop count and flip one of their incoming links (in red) so that they can route to their children and increase the number of paths to the sink. Now, no nodes have a hop count of 1 and we have a routing loop. Let's assume nodes 1 and 2 increment their hop count to 4 to ensure they have a larger hop count than their parents. Nodes 3 and 4 will then do the same. The process repeats and the hop counts will "count to infinity."

one node with hop count h before the packet can reach the sink, regardless of the actual routing topology used to route the packet.

Therefore, a natural approach is to start with a minimum hop DAG and modify it to get a robust routing topology. For instance, we may want to *demote* a node, meaning increase its hop count from the minimum, so that it may route a packet to the root through one of its children (higher hop count) or one of its siblings (same hop count). However, the process of modifying a minimum hop DAG to get a *robust DAG* is tricky because a slight miscoordination between nodes can result in the formation of routing loops in the topology.

Consider the example illustrated in Figure 2.23, which is similar to the well known *count-to-infinity* problem in networking. Here, we assume that nodes with hop count h can only route to nodes with hop count $h' < h$. As such, if a node wishes to route to its children to increase its number of paths to the root, it must increase its hop count. If two such nodes do this simultaneously, nodes 3 and 4 in the example, we can get inconsistent hop counts and routing loops.

Instead of trying to form a robust DAG by demoting nodes in a minimum hop count DAG, we will directly assign nodes a *robust count* $\hbar_i$ for each node $i = 1, \ldots, N$ following Algorithm 5. $\boldsymbol{\hbar}$ denotes the vector of assigned hop counts $[\hbar_1\ \hbar_2\ \cdots\ \hbar_N]$. The robust hop count will always be greater than or equal to the minimum hop count of a node, and always less than or equal to the maximum hop count. Algorithm 5 uses a variant of the robustness metric, $\overleftarrow{r}$, to help assign hop counts to nodes, assuming nodes can only route to other nodes with smaller (or equal) hop counts. Let u_i be a downstream neighbor of node v,

unlike (2.5) where u_i are the upstream neighbors of v. Then,

$$\begin{aligned} \overleftarrow{r}_{b\to b} &= 1 \\ \overleftarrow{r}_{v\to b} &= 1 - \prod_i (1 - p_{vu_i} \overleftarrow{r}_{u_i\to b}) \quad . \end{aligned} \tag{2.17}$$

Note that this definition of $\overleftarrow{r}$ is easy to compute during topology formation with very little communication between nodes. In fact, $\overleftarrow{r}_{v\to b}$ on the topology G is equal to $r_{b\to v}$ on the topology $\overleftarrow{G}$, formed by reversing the orientations of all the links in G.

Algorithm 5 ROBUST_HOP_COUNT

Input: $G = (\mathcal{V}, \mathcal{E}, p), a, b, \boldsymbol{\tau}, K$

Output: $\boldsymbol{\hbar}$ ▷ Robust hop count.

$\forall i, \hbar_i = \texttt{NIL}$ ▷ `NIL` means not yet assigned.

Select a set of nodes $\mathcal{C}$ connected to a with good links, and $\forall v \in \mathcal{C}, \hbar_v := 1$.

for $k := 1$ to K **do**

  for all $v \in \{v \in \mathcal{V} : \hbar_v = \texttt{NIL}\}$ **do**

    [Run the following simultaneously on all nodes]

    for $h := 1$ to $\max_i \hbar_i$ **do**

      Let $u_i \in \mathcal{V}$ be nodes with $\hbar_{u_i} < h$ and a link to v.

      $\overleftarrow{r}_{v\to b} := 1 - \prod_i (1 - p_{vu_i} \overleftarrow{r}_{u_i\to b})$

      if $\overleftarrow{r}_{v\to b} \geq \tau_{k-h+1}$ **then**

        $\hbar_v := h$

        Break from innermost **For** loop.

      end if

    end for

  end for

end for

Return: $\boldsymbol{\hbar}$

Algorithm 5 joins nodes to the robust routing topology (by assigning them a robust hop count $\hbar$) over rounds $1, \ldots, K$, where each round lasts a fixed interval of time. At each round k, node v has the potential to be assigned one of several robust hop counts. If node

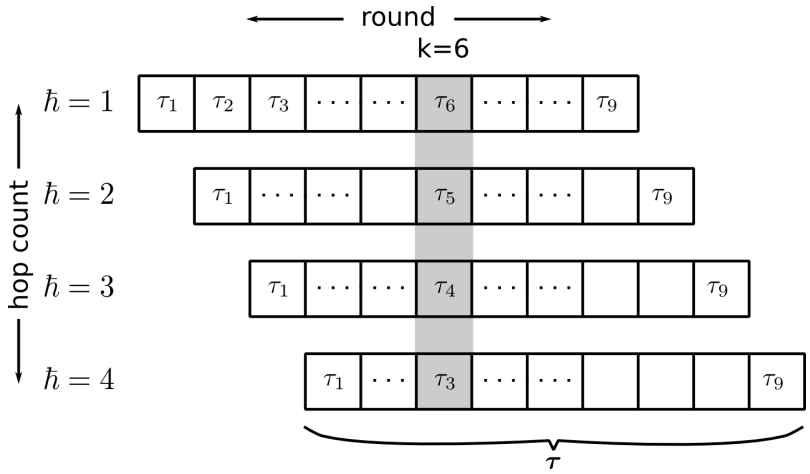

Figure 2.24. Illustration of how thresholds are used to help assign a node a robust hop count. The horizontal row of thresholds represent $\boldsymbol{\tau}$, with $\tau_E = \tau_9$, which is some value less than the minimum link probability in G. The shaded vertical column of thresholds are the thresholds tested by a node at round k. A node v picks the smallest hop count $\hbar$ such that $\overleftarrow{r}_{v \to b} \geq \tau_e$.

v is assigned a robust hop count $\hbar$, then $\overleftarrow{r}_{v \to b}$ from (2.17) must be greater than or equal to a threshold $\tau_e \in [0, 1]$. The downstream neighbors u_i used to compute $\overleftarrow{r}_{v \to b}$ must have been assigned a robust hop count less than $\hbar$ in a round before round k. The value of the threshold τ_e corresponding to a particular hop count $\hbar$ depends on the *epoch* $e = k - \hbar + 1$. Note that to seed the algorithm, a set of nodes with good links to the sink are assigned $\hbar = 1$ before round 1.

In round k, for each possible $\hbar$, node v computes the corresponding $\overleftarrow{r}_{v \to b}$ and compares it against the corresponding threshold τ_e. If v can be assigned multiple $\hbar$ (there are multiple $\hbar$ such that $\overleftarrow{r}_{v \to b} \geq \tau_e$), v will choose the smallest $\hbar$. The robust hop count assigned to v is denoted $\hbar_v$, and remains fixed thereafter. The vector of thresholds is denoted $\boldsymbol{\tau} = [\tau_1 \ \cdots \ \tau_e \ \cdots \ \tau_E]$. The thresholds decrease with increasing e, with the last threshold τ_E less than the minimum link probability in G. This is illustrated in Figure 2.24.

Although Algorithm 5 is written as sequential pseudocode, it is meant to be implemented in parallel on the nodes in the network. First, all the nodes are time synchronized (by some sort of broadcast algorithm, for instance [71]) so they share a global time t, used to keep track of the current round k. Also, all the nodes have the vector of thresholds $\boldsymbol{\tau}$. In each round, each node v listens for a broadcast of the pair $(\overleftarrow{r}_{u_i \to b}, \hbar_{u_i})$ from each of its neighbors u_i that have joined the routing topology. After receiving the broadcasts, node v performs

the computations and comparisons with the thresholds to determine if it can join the routing topology with some robust hop count $\hbar_v$. Once v joins the network, it broadcasts its value of $(\overleftarrow{r}_{v\to b}, \hbar_v)$.

Algorithm 5 is designed to reduce the chance of forming routing loops despite packet losses. For instance, nodes can periodically broadcast their own $\overleftarrow{r}$ and $\hbar$. If a broadcast of $(\overleftarrow{r}_u, \hbar_u)$ from node u is missed by node v, then node v may join the network later or join with a higher hop count (because of a lower threshold when joining the network later). Under the rule that nodes must route to nodes with a lower hop count, this should not cause a routing loop. In fact, we can add edges to $\mathcal{E}$ (channel conditions improve between two nodes) or nodes to $\mathcal{V}$ (and their adjacent undirected edges to $\mathcal{E}$) in the middle of assigning robust hop counts and still not get routing loops in the topology. Problems arise if all edges from a node to nodes with a lower hop count are removed from the topology, but that should occur infrequently.

If we orient the edges in $\mathcal{E}$ such that nodes with higher hop counts transmit to nodes with lower hop counts, we get a robust DAG. However, it will not utilize edges between nodes with the same hop count. The remedy is to use a simple mechanism to create a DAG on just the nodes with hop count $\hbar$ and the edges between them. For instance, you can assign the node v a *robust fractional hop count*

$$\hbar^f = \hbar + (1 - \overleftarrow{r}_{v\to b}) \tag{2.18}$$

Similarly, for a minimum hop DAG to use links between nodes with the same hop count h, you can assign v a *fractional hop count*

$$h^f = h + (1 - p_{\max}) \tag{2.19}$$

where $p_{\max}$ is the largest link probability between v and nodes with a lower hop count. Both mechanisms require no communication between nodes. Then, we orient the edges from nodes with higher (robust) fractional hop counts to nodes with lower (robust) fractional hop counts. In the rare case of a tie, we can use the node id as the tie breaker.

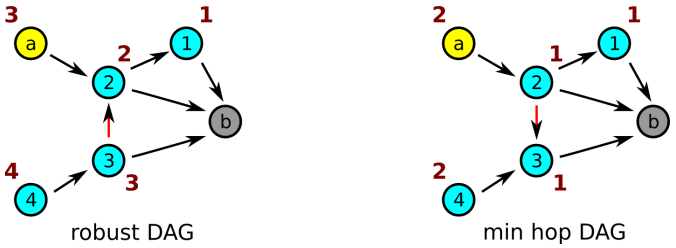

Figure 2.25. An example where the minimum hop DAG provides more paths between a and b than the robust DAG. Nodes are labeled by their hop count (not encircled), where we chose one possible set of hop assignments for the robust DAG. Let's assume the nodes join the network in the order of their node ids and node a joins last. In the robust DAG, node 3 may wish to connect as a child of node 2 to get more paths to the root for itself (and for node 4), which results in less paths from a to b.

2.4.3 Example and Discussion

Note that a robust DAG generally tries to maintain a high $\overleftarrow{r}$ between every node and the sink, not a particular source-destination pair. Therefore, there may be scenarios where a given source-destination pair has a higher $\overleftarrow{r}$ on a minimum hop DAG than on a robust DAG, as shown in Figure 2.25.

Example 2.4.1 (Robust topology on random connectivity graphs). This example uses the robustness metric and path probability metric to compare robust topologies with minimum hop topologies on randomly generated, undirected connectivity graphs. Both types of routing topologies are rooted at the sink, and we compare the connectivity of all the nodes to the sink. The robust topology was generated from the robust fractional hop counts $\hbar^f$ returned by Algorithm 5 and (2.18). The threshold vector used in the algorithm was $\boldsymbol{\tau} = [0.99\ 0.98\ \cdots\ 0.6]$ where thresholds in consecutive epochs differed by 0.01. The maximum round to join the network was $K = 100$. The minimum hop topology was generated from the fractional hop counts h^f returned by (2.19).

The undirected connectivity graphs were generated by placing 30 nodes uniformly at random in a 10×10 grid. To prevent extremely dense clusters of nodes from forming (making it hard to visualize the results), if a node was placed too close (less than 0.5 units) away from another node it was removed and placed again. To ensure that the graph was reasonably well connected, we only used randomly generated graphs where the node in the

Table 2.1. Robust DAG vs. min hop DAG on 50 random graphs

Routing Topology	Robustness Metric $r_{v\to b}$ mean	median	variance	Path Prob Metric $p_{v\to b}$ mean	median	variance
Min Hop	0.9126	0.9215	0.0038	0.8544	0.8740	0.0100
Robust	0.9227	0.9275	0.0025	0.8707	0.8870	0.0074

lower left corner was connected by an undirected path to the node in the upper right corner. The node in the upper right corner was designated as the sink in the network.

Nodes that were less than 2 units apart had a link, nodes more than 3 units apart did not have a link, while nodes that were between 2 and 3 units apart had a link with probability $3 - d$, where d was the Euclidean distance between the nodes. Links were then assigned a probability uniformly at random from the range $[0.7, 1]$, independent of the distance between the nodes.

Figure 2.26 is an example of a randomly generated connectivity graph, the robust and minimum hop routing topologies constructed on the graph, and the robustness and path probability metrics $r_{v\to b}$ and $p_{v\to b}$ on the two routing topologies (NOT the robustness metric variant $\overleftarrow{r}_{v\to b}$).

Figure 2.27 compares the connectivity of a robust topology with a minimum hop topology on 25 randomly generated undirected connectivity graphs. The comparison is made on nodes that have joined both topologies (e.g., we don't include nodes that have joined the minimum hop topology but have not joined the robust topology in our comparisons). The distribution of connectivity metrics on all nodes in each graph is represented by a box and whiskers plot. Comparing the respective box and whiskers plot for each run (same columns in the top and bottom graphs), we see that the robust DAG tends to give more nodes better connectivity to the sink, whether we measure connectivity by the robustness metric or the path probability metric. Table 2.1 shows the aggregate results on 50 random graphs (including the 25 in Figure 2.27). This shows that the robust DAG does tend to produce better routing topologies when the topology is shared by many source nodes in the network routing to the sink.

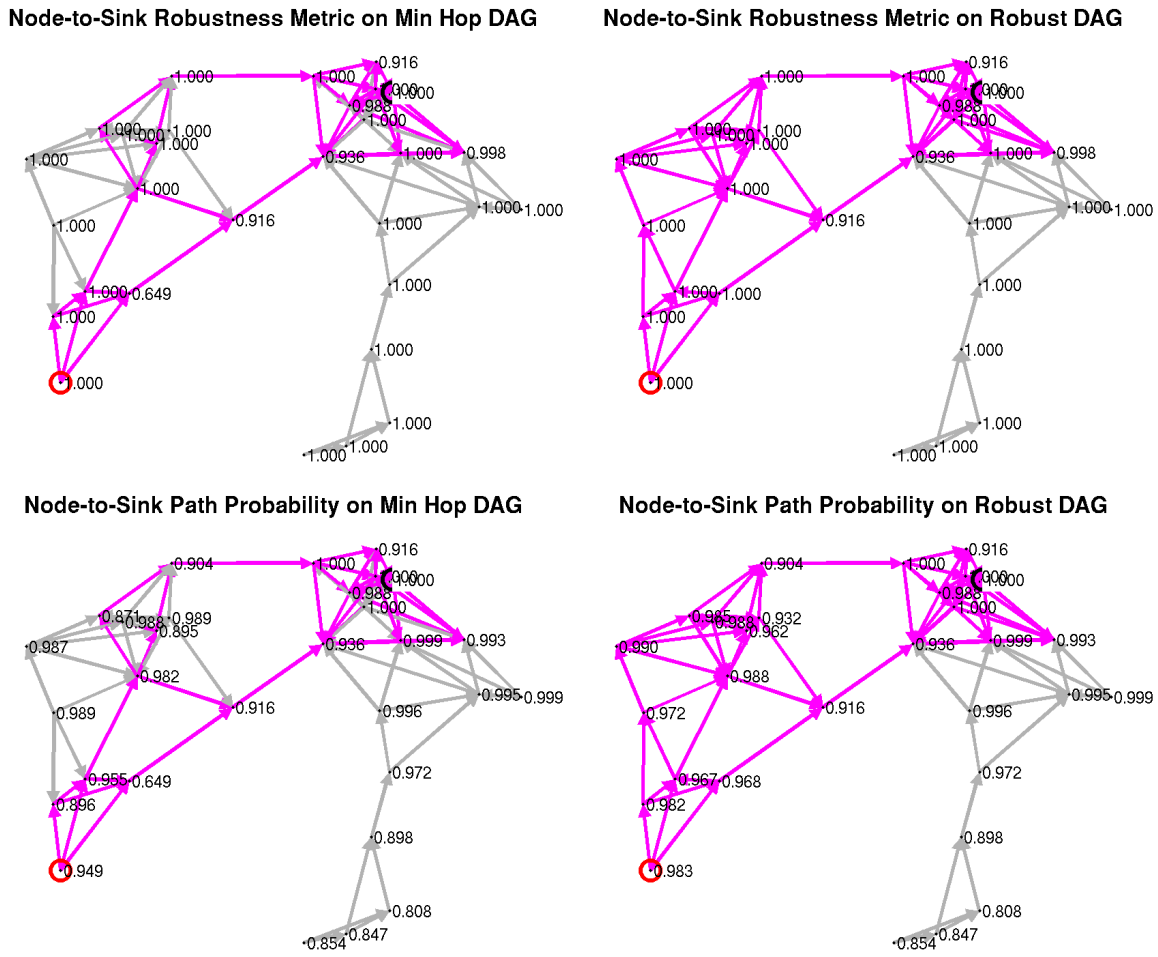

Figure 2.26. Path Probability and Robustness Metric comparisons of the Robust DAG and Min Hop DAG on a randomly generated topology with 30 nodes. The source node is circled in red, The sink node is circled in black, and links on the path between the source and the sink are magenta (the remaining links are gray). Note that all nodes can route to the sink in this topology, but a source was selected to highlight the set of paths to the sink from one node. Nodes are labeled with the metric $p_{v\to b}$ or $r_{v\to b}$ on the respective plots. Some metrics are displayed as 1 due to roundoff errors (only the sink actually has a metric equal to 1).

□

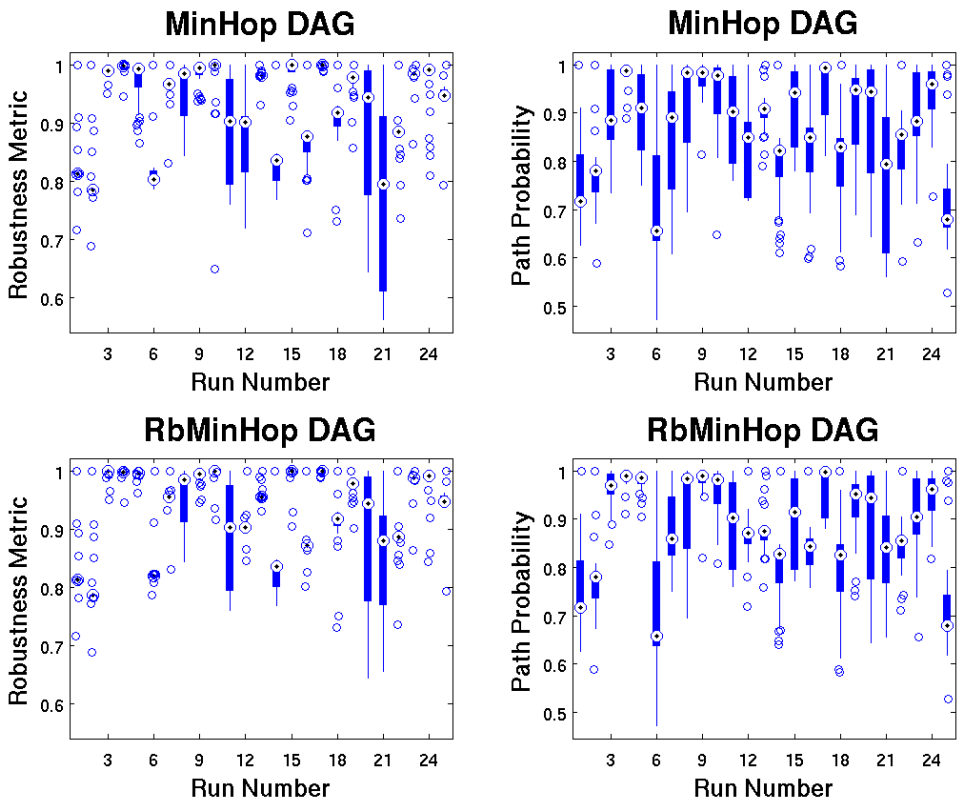

Figure 2.27. Comparison of routing topologies generated by the robust DAG and a minimum hop DAG, using the robustness metric and the path probability metric. The distribution of metrics from each node to the sink for is represented by a box and whiskers plot. The median is represented by a circled black dot, the outliers are represented by blue circles, and the interquartile range (IQR) is 1.5. See text for details.

Chapter 3

Metrics for Scheduling Transmissions

The problem of scheduling links in a TDMA network for transmission has been studied extensively and is known to be a hard problem [84; 83]. This chapter is concerned with finding a tractable, distributed scheduling algorithm for wireless TDMA mesh networks. Ideally, we would want a schedule that would optimize metrics to better support real-time control systems over wireless sensor networks. The types of metrics for this will be discussed in detail in Section 3.2.2.

We are interested in *distributed* scheduling algorithms primarily because they have the potential to be more adaptable to changing wireless channel conditions than centralized scheduling algorithms. A centralized scheduling algorithm needs to collect information from all the nodes and then distribute a schedule to all the nodes. This means one *root* node (or multiple root nodes, if they are connected by a wired backbone) will become a communication bottleneck just for *network control traffic*, the traffic that is used to support the networking algorithms. Recall that wireless communications in WSNs is fairly slow, which means that the time spent passing information to and disseminating information from the root node will be a significant fraction of the total time to compute and install a new schedule in the network. If this root node is also the collection point for data in the

network (e.g., many-to-one routing), the network control traffic will also reduce the data throughput in the network.

The tradeoff of using a distributed scheduling algorithm is that it computes less optimal schedules than a centralized algorithm. However, in practice centralized algorithms may also compute suboptimal schedules because optimal schedules often require a long time to compute. We are not as interested in finding optimal schedules as finding schedules that perform well and can be computed with reasonable costs, i.e., reasonable time and communication overhead. The next section will first review the relevant literature on scheduling and the following sections will describe a scheduling algorithm tailored for reliable sensor networks.

3.1 Background and Related Work on Scheduling

The objective of TDMA scheduling is to find sets of links that can transmit simultaneously in a network in order to optimize a metric such as network throughput or latency. This problem fits under a unified problem framework for TDMA / FDMA / CDMA[1] channel assignment proposed by Ramanathan [84]. Our problem is defined by the *constraints* of which transmitters and receivers can be active at any moment in time on a particular frequency channel.

Two common interference models used to define transmission constraints are the *node-exclusive* interference model and the *IEEE 802.11 DCF* interference model. In the node-exclusive interference model, a node can only transmit or receive from one other node at a time. This model applies to networks where each node has only one radio and neighboring links do not interfere with each other, for example frequency hopping networks such as Bluetooth and FH-CDMA networks. The 802.11 Distributed Coordination Function (DCF) interference model has the same constraint as the node-exclusive model plus the additional constraint that links must be at least two hops apart. This is because any transmission on a link must undergo a successful Request to Send / Clear to Send (RTS / CTS) handshake,

[1]Time / Frequency / Code Division Multiple Access.

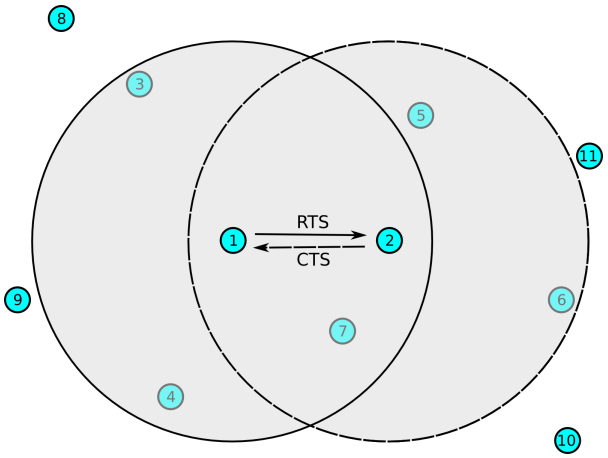

Figure 3.1. Illustration of a RTS / CTS handshake for 802.11 DCF, used to prevent the *hidden terminal* problem. The transmission range (large circle) is assumed to be the same as the interference range. A transmitter (node 1) broadcasts an RTS packet designating the intended receiver (node 2), which preempts all its neighboring nodes from sending (nodes 3, 4, and 7). Upon receiving an RTS packet, the receiver responds with a CTS packet only if it has not received an RTS or CTS packet from its neighboring nodes (nodes 5, 6, and 7). This CTS packet preempts neighboring nodes from sending an RTS packet. The transmitter, upon receiving the CTS packet, can start transmitting the (longer) data packet. In the illustration, all nodes in the shaded circles do not transmit or receive during the data packet transmission. RTS / CTS is typically not used in sensor networks because the data packets are small.

as described in Figure 3.1. The node-exclusive and 802.11 DCF interference models can be generalized to k-hop interference models.

Definition 3.1.1 (k-hop Interference Model)**.** If link (u_1, v_1) is scheduled, then any other link (u_2, v_2) that is simultaneously scheduled must satisfy

$$\min_{\substack{x\in\{u_1,v_1\}\\ y\in\{u_2,v_2\}}} h(x,y) \geq k \quad , \tag{3.1}$$

where $h(\cdot,\cdot)$ measures the minimum number of hops between two nodes. □

Thus, the node-exclusive interference model is a 1-hop interference model while an 802.11 DCF interference model is a 2-hop interference model.

Alternatively, the interference constraints between nodes can be defined based on other measures such as the geometric distance between nodes. The advantage of defining an interference model based on the hop distance between nodes is that this doesn't impose idealized radio propagation models (e.g., disk coverage models) on the nodes in the network

and this corresponds to information that is readily available to the nodes. For instance, node i may have difficulty directly estimating which nodes' transmissions are interfering with node i's reception because node i may not be able to decode the contents of the interfering messages. The drawback is that k-hop interference may not accurately model the actual interference in the environment.

The scheduling problem is closely related to the graph coloring, independent set, and matching problems on *undirected graphs*. The following subsections will introduce these problems, describe existing solutions to the *distributed* scheduling problem, and review the current understanding of how scheduling at the logical / data-link layer affects the network and transport layers in the network.

3.1.1 Coloring, Independent Sets, and Matching

The usual approach to solving the scheduling problem is to translate it to the graph coloring, matching, or independent set problem and then apply an algorithm to solve the new problem. First, let us define some relevant terminology.

Definition 3.1.2 (Graph Coloring [44])**.** A *coloring* of a graph $G = (\mathcal{V}, \mathcal{E})$ is a map from $\mathcal{V}$ into a finite set of colors such that no two adjacent vertices are assigned the same color (See Figure 3.2, left). If G can be colored with a set of k colors, then we say that G can be k-colored. The smallest value of k for which G can be colored is the *chromatic number* of G, and is denoted by χ_G. The set of vertices with a particular color is called a *color class* of the coloring. □

A color class of a coloring is also an *independent set*. Note that we often use positive integers as "colors", so nodes are labeled by integers when assigned to a color class.

Definition 3.1.3 (Independent Set [44])**.** An *independent set* of vertices in a graph is a set of vertices such that no two vertices in the set are adjacent to each other. A *maximal independent set* (MIS) is an independent set that is not a subset of any other independent set (See Figure 3.2, middle). A *maximum independent set* is an independent set with the most number of vertices of all independent sets on the graph. □

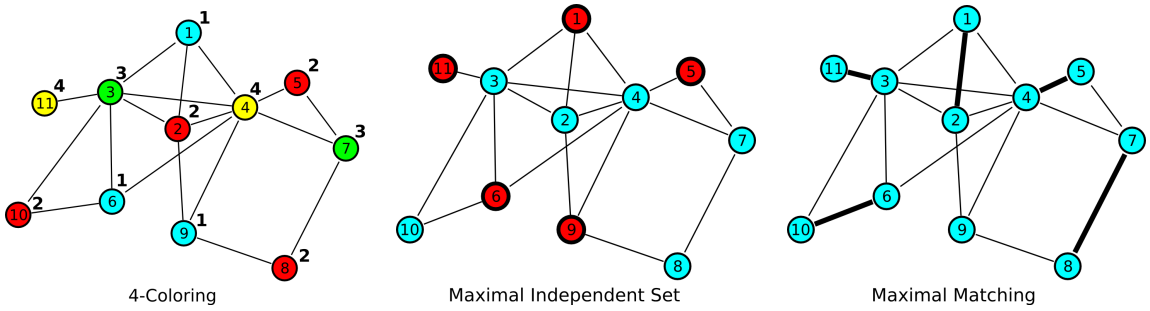

Figure 3.2. (left) Nodes in the 4-coloring are labeled by their color. (middle) Nodes in the maximal independent set are bold. (right) Edges in the maximal matching are bold.

Definition 3.1.4 (Matching [44])**.** A (2-D) *matching* M in a graph is a set of edges such that no two edges have a vertex in common. A *maximal matching* is a matching that is not a subset of any other matching (See Figure 3.2, right). A *maximum matching* is a matching with the most number of edges of all matchings on the graph. □

The problem of scheduling broadcasts can be translated to coloring a graph. The problem of scheduling links (particularly for the node-exclusive interference model) can be translated to finding maximal matchings. These problems, as well as the problem of finding maximal independent sets, are all closely related and will be discussed later.

First, let us walk through a simple example of how to translate a broadcast scheduling problem to a graph coloring problem. Suppose our problem is to find a minimum length schedule of broadcasts where every node gets to transmit, because a shorter schedule can give better network throughput. The constraints are defined by the 2-hop interference model, meaning that a node can only receive from one other node in a time slot and a node cannot transmit in the same time slot as another node within two hops away. Our "Minimum length" objective translates to a "minimum colors" objective in the corresponding graph coloring problem. First, we create an undirected graph C that embodies the constraints of the scheduling problem (See Figure 3.3). Start with the routing topology G, and replace each directed edge with an undirected edge. Then, add edges between nodes that are two hops away to get C. Next, we color C and assign nodes with the same color to the same time slot for transmission.

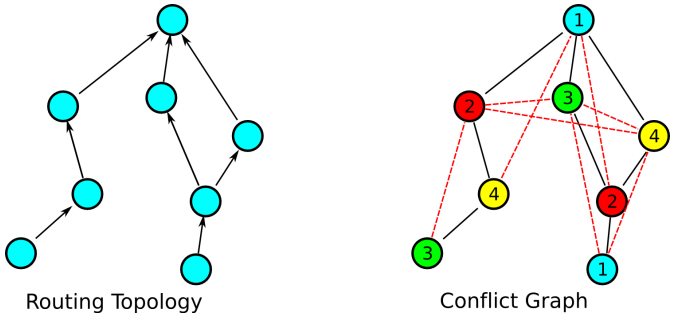

Figure 3.3. (left) Routing topology G. (right) The conflict graph C associated with G, with colored nodes. The dashed lines represent 2-hop edges added to the conflict graph.

The general approach for *reducing* a scheduling problem to a graph problem is to create a graph C from G such that the edges of C represent *transmission-reception constraints* (e.g., a node can only receive from one other node at a time) and *interference constraints* (e.g., a nearby node cannot be simultaneously transmitting) on the new problem [83]. In the example above, the interference constraint was defined by the number of hops separating two transmitters. The coloring problem with a k-hop separation requirement on G is called *distance-k* coloring, so the example above was an example of a *distance-2* coloring problem on G. In general, C can represent arbitrary interference constraints between nodes. For more examples of constraint graphs for different scheduling problems, see [31].

Unfortunately, the problem of coloring a graph with the minimum number of colors is an NP-complete problem (for a discussion on NP-completeness, see [20]). Since graph coloring on arbitrary graphs is NP-complete, one might look into graph coloring on restricted classes of graphs. Unfortunately, graph coloring on even very restrictive classes such as *planar-point graphs* [93] and *unit disk graphs* [38] have been shown to be NP-hard.

Definition 3.1.5 (Planar-Point Graph and Unit Disk Graph [93; 38])**.** Given a set of points $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$ in an m-dimensional space, each specified by an m-dimensional vector $[x_1 \cdots x_m]$, and a range r_i associated with each $\boldsymbol{x}^{(i)}$, the *point graph* constructed from this set of points is a graph $G = (\mathcal{V}, \mathcal{E})$, where each node v_i represents a point $\boldsymbol{x}^{(i)}$ and there is a directed edge from node v_i to v_j if $\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2 \leq r_i$, where $\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2$ is the Euclidean distance between the points $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$. *Planar point graphs* are point

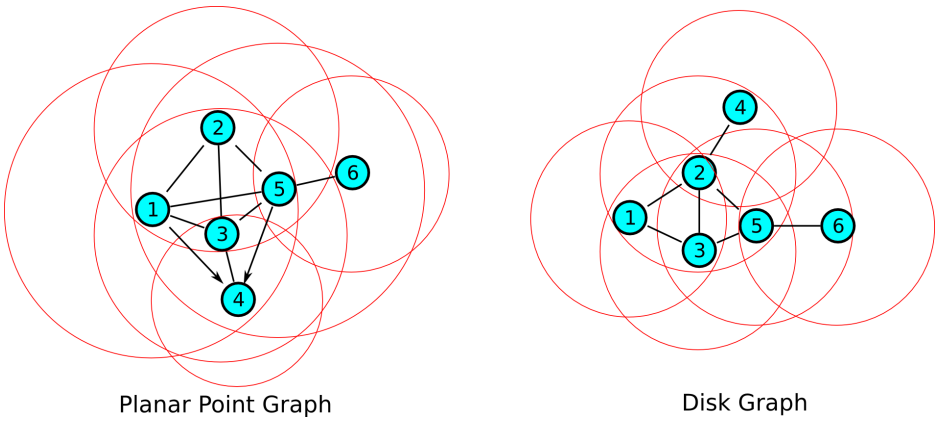

Figure 3.4. (left) A planar-point graph. (right) A disk graph. For clarity, we represent two directed edges (i, j) and (j, i) as one undirected edge.

graphs constructed from the points on a 2-dimensional plane ($m = 2$). A *unit disk graph* is a planar point graph where $\forall i, r_i = 1$. See Figure 3.4 for examples. □

On the other hand, the problems of finding a maximal matching and a maximal independent set are *not* NP-complete. In fact, both can be solved by a distributed algorithm described in the next subsection. A maximal matching on G gives a set of links that can transmit simultaneously in one time slot under the node-exclusive interference model. A maximal independent set on G allows us to compute a suboptimal coloring in a randomized, distributed manner, as we will see in Section 3.1.2. Also, a maximal matching problem can be *reduced* to a maximal independent set problem by converting the graph G to it's *line graph* $L(G)$, where each edge in G is represented by a vertex and two edges in G are represented by adjacent vertices in $L(G)$ if and only if they are incident (share a node) in G [44].

The related problems of finding a *maximum* matching and a *maximum* independent set are more difficult. There are algorithms to find a maximum matching on a graph in polynomial time ($O(N^3)$ where N is the number of nodes, [77]). However, the problem of finding the maximum independent set of a graph is NP-complete [77]. Furthermore, variations on the maximum matching problem such as the *3-dimensional matching problem* are NP-hard [77]. A variant that is relevant to scheduling is the class of *maximum weighted k-valid matching problems*. In these problems, the edges have weights and the objective

is to find a matching that respects the k-hop interference constraints and maximizes the sum of the weights of edges in the matching. A matching satisfying the k-hop interference constraint, called a k-valid matching, does not contain edges that are within k-hops of each other, as defined by (3.1). The maximum weighted k-valid matching problems are also NP-hard for $k \geq 2$ [97].

Naturally, when problems are NP-hard we look for approximate solutions that are more computationally tractable. Some NP-hard problems are amenable to *polynomial-time approximation schemes*.

Definition 3.1.6 (Polynomial-time Approximation Scheme (PTAS) [77])**.** An algorithm is a *polynomial-time approximation scheme* (PTAS) for an optimization problem A if, when supplied with an instance of A and an $\epsilon > 0$, it returns an ϵ-approximate solution (i.e., solution within $(1 + \epsilon)$ times the optimal) within time which is bounded by a polynomial (depending on ϵ) in n. □

Although such approximation schemes are highly desirable, not all problems are PTAS. For instance, if P $\neq$ NP, then it has been shown by [2] that the chromatic number χ_G can not be computed within a factor of $|\mathcal{V}|^{1/7-\epsilon}$ for all $\epsilon > 0$ in polynomial time. On the other hand, the maximum weighted k-valid matching problems have a PTAS with an approximation factor of $1/(1 + \epsilon)$ if we restrict our attention to unit disk graphs [97]. In fact, a number of problems such as maximum independent set and vertex coloring have been shown by [68] to be approximable within a constant factor, but not for *all* constant factors $\epsilon > 0$ (as in a PTAS), on unit disk graphs.

While maximum matchings and minimum coloring help compute schedules that maximize throughput, the above results show that they may be difficult to implement. In fact, finding a maximal matching for scheduling may be "good enough" in practice. Also, we shall see that maximal matching is very suitable for distributed implementation.

3.1.2 Complexity of Distributed Scheduling

Distributed scheduling tries to exploit the locality of scheduling constraints to reduce the amount of coordination between nodes and allow computations to happen in parallel. Under the *synchronous* model of computation, the nodes share a global clock, the link delays are bounded, and computation happens in rounds alternating with link transmissions [79]. The challenges in distributed computing include minimizing the amount of information exchanged between nodes and minimizing the number of rounds of computation, while computing a "reasonably good" answer. Other concerns are whether the computed solution is robust to node or link failure.

For comparison with parallel algorithms later, let us consider a simple, *sequential* (not parallel), greedy algorithm called GREEDY_COLOR which finds a $(\Delta + 1)$-coloring of a graph for TDMA scheduling in $O(|\mathcal{E}|)$ time, where Δ is the maximum degree of the undirected connectivity graph. Sequential algorithms assume that only one node performs a computation and transmits the results at a time, so they result in many rounds of computation. GREEDY_COLOR selects a node (perhaps through lexicographic ordering of the node id) and colors the node with the smallest positive integer not in use by its neighbors. GREEDY_COLOR uses at most six times more colors than an optimal coloring when we restrict ourselves to coloring unit disk graphs [68]. Variants of GREEDY_COLOR select the nodes in different orders. For instance, [84] describes RAND, MNF, and PMNF coloring where nodes are labeled with an ordering and then colored with GREEDY_COLOR in *decreasing* order of the labels. RAND labels the nodes randomly, while Minimum Neighbors First (MNF) assigns smaller labels to nodes with fewer neighbors so that nodes with a higher number of neighbors are colored first. Progressive Minimum Neighbors First (PMNF) also assigns smaller labels to nodes with fewer neighbors, except that as a node is labeled (nodes with fewer neighbors are labeled first), the node and its incident edges are removed from the graph. Ramanathan showed that in practice, these variants have different performances.

If we wish to limit the number of rounds of computation, we will need to run the algorithm in parallel on multiple nodes. If we are given a maximal independent set $\mathcal{L}$ in the

network, we can use the nodes in $\mathcal{L}$ as the *leaders* of local clusters to coordinate parallel computations. An MIS gives a set of cluster leaders that are not adjacent to each other and ensures that each node in the network is adjacent to at least one cluster leader. This stage of the algorithm needs to use CSMA/CD to exchange information between nodes because we do not yet have a TDMA schedule (a bootstrapping problem). Herman et al. [42] give a distributed algorithm which uses a maximal independent set $\mathcal{L}$ as the leader nodes to perform a (suboptimal) distance-2 coloring of the network. A brief description of the algorithm, which is run in parallel on the nodes, is provided in Algorithm 6.

Algorithm 6 Distributed MIS-based Coloring Algorithm [42]

Let $\mathcal{N}_i$ be the set of nodes adjacent to i, and $\mathcal{N}_i^2$ be the set of nodes within two hops of i. (similarly for $\mathcal{N}_j, \mathcal{N}_j^2$)

Code for $i \in \mathcal{L}$

In Round 1 **do**:

i assigns a minimum number of unique colors to all the nodes in $\mathcal{N}_i \cup i$ and broadcasts this color assignment to $\mathcal{N}_i$.

On receiving color-leader pairs for $\mathcal{N}_i^2$ (Rounds 2, 3,...) **do**:

i reassigns colors to nodes in $\mathcal{N}_i \cup i$ assuming that the color assignments to nodes in $\mathcal{N}_i^2$ by leaders with a lower node id have priority and will not be changed. This new color assignment will use a minimum number of colors and follow the constraints of distance-2 coloring on the nodes in $\mathcal{N}_i \cup i$ and nodes in $\mathcal{N}_i^2$ colored by higher priority leaders.

i broadcasts this color assignment to $\mathcal{N}_i$.

Code for $j \notin \mathcal{L}$

After j receives its color assignments from neighboring leader nodes i, it accepts the color assignment c_j from the leader node with the lowest node id, $i_{\min}$.

j broadcasts the color-leader pair $(c_j, i_{\min})$ to nodes in $\mathcal{N}_j$

After hearing from all nodes in $\mathcal{N}_j$, node j broadcasts the set of color-leader pairs from $\mathcal{N}_j$ to node j's leader nodes i. Node i now has all the color-leader pairs from its two hop neighborhood, $\mathcal{N}_i^2$.

In practice, we expect this algorithm to run much faster than a serial algorithm on most networks, although the number of communication rounds before the network converges depends on how long it takes to propagate a message across the network. Also, we may be able to use the partially computed coloring before the algorithm converges to create a broadcast schedule with a small number of collisions.

A simple, parallel, *randomized* algorithm for computing an MIS was given in [66] with polylogarithmic expected running time $EO(\log^2 N)$. At each round, a node i decides whether to include itself in the MIS with a given probability $p_i = \frac{1}{2\delta(i)}$, where $\delta(i)$ is the degree of node i. If one of its neighboring nodes with higher degree also decides to join the MIS in the same round, node i will remove itself from the MIS. The choice of p_i is critical to the analysis of the running time for this algorithm.

Using a distributed algorithm to compute an MIS, we can also compute a $(\Delta + 1)$-coloring of the network in a distributed manner by using a reduction of the $(\Delta+1)$-coloring problem to the MIS problem [64; 54]. Given a graph G, we first construct a graph G' as follows. For each node i, create $\Delta + 1$ copies $i_0, \ldots, i_\Delta$ and connect these nodes to form a $(\Delta + 1)$-clique. Next, for each $x \in \{0, \ldots, \Delta\}$, connect two nodes i_x and j_x (note the same subscript x) if G contains an edge (i, j). If we find the MIS on G', we can convert the solution to a $(\Delta + 1)$-coloring on G by just coloring the node i with the color x if i_x is in the MIS. Recall that it is impossible for two nodes i_x and i_y to be in the MIS if $x \neq y$ because they are both in the clique representing node i. Note that the MIS problem on G' can be solved using a distributed algorithm on the nodes in G by "simulating" on node i each clique of nodes representing node i.

We close this subsection by noting that just as the $(\Delta + 1)$-coloring problem can be reduced to an MIS problem, we can also use the Distributed Color2MIS algorithm, Algorithm 7, to quickly find the MIS given a graph coloring.

Algorithm 7 Distributed COLOR2MIS [79]

Input: $G = (\mathcal{V}, \mathcal{E}), \{\mathcal{C}_c\}_{c=1,\ldots,m}$ ▷ $\mathcal{C}_c$ is a color class.

Output: $\mathcal{L}$ ▷ $\mathcal{L}$ is the MIS.

for $c := 1$ to m **do** ▷ One round of computation.

[**Run the following simultaneously on all nodes** v]

if $v \in \mathcal{C}_c$ **then**

if none of neighbors $\mathcal{N}_v$ joined $\mathcal{L}$ **then**

$\mathcal{L} := \mathcal{L} \cup \{v\}$

Broadcast $v \in \mathcal{L}$ to all neighbors $\mathcal{N}_v$.

else

Decide that $v \notin \mathcal{L}$.

end if

end if

end for

3.2 The Scheduling Problem Description

The scheduling problem on wireless sensor networks has the same objectives and constraints as the scheduling problem on packet radio networks, with the addition of resource constraints on each node in the network. Nodes have limited memory, computational power, energy supply, and bandwidth. Instead of solving for an optimal schedule to use on the network, which can be both computationally expensive and difficult to implement on the resource constrained nodes, the scheduling algorithm described in Section 3.3 employs distributed mechanisms that tend to benefit our objectives (whether by increasing or decreasing them), but may not attain the optimal values for our objectives.

Our scheduling algorithm will generate a schedule for Unicast Path Diversity (UPD, described in Section 4.2.1), meaning a single copy of a packet will be delivered through the network. This section presents the scheduling objectives and constraints, and motivates their importance through examples of schedules that do not take them into consideration.

We first introduce the interference model since this, more than the objectives, is what defines a wireless scheduling problem.

3.2.1 Scheduling Interference Model

As mentioned in Section 3.1.1, the interference model has a large effect on the complexity of the scheduling problem. We will use the 1-hop (node-exclusive) interference model and assume that nodes have one radio which cannot simultaneously receive and transmit. This model is valid on 802.15.4 networks if interfering nodes transmit on different frequencies and there are enough frequency channels (16 channels in the 2.4 GHz band) for all interfering nodes to get a unique channel. The latter is satisfied if the network density is low or if nodes in the network are selected to sleep and not transmit if the density is too high. In effect, we have split the scheduling problem into two subproblems: finding a (maximal) matching and then coloring the links with frequencies, assuming the link interference graph is 16-colorable.

One difficulty is that the interference graph is not always known to the network, even with global information from all the nodes. This is because the transmission from node a can interfere with the reception at node b, but node b may not be able to decode the packet from node a to know that node a is interfering, or even that the interference is not from a source outside the network. It is well known that the wireless interference range can be larger than the transmission range. There is no easy solution to this problem. Preferably, any interference from nodes in the network whose source cannot be identified will happen at random intervals and can be treated as random outside interference. This, along with other considerations described below, supports the use of some randomization in our scheduling algorithm.

3.2.2 Scheduling Objectives

The objectives of our scheduling algorithm are to

- reduce the *latency* of end-to-end packet delivery,

- increase the *reliability* of end-to-end packet delivery, and
- reduce the *association* (correlation) of consecutive packet delivery events (reduce the number of consecutive packet drops).

Reducing the queuing of packets in the network will reduce latency. Reliability is partially addressed by using a good routing topology, but also depends on how the links are scheduled and whether packets are likely to be trapped at nodes in the network whose outgoing links have low probability (see Section 2.3). The association between consecutive packet delivery events is lowered if the paths traversed by consecutive packets share very few links and nodes. Lowering this will reduce the likelihood of having many consecutive packet drops, which are especially detrimental to the stability and performance of real-time control systems. The three objectives of reducing latency, increasing reliability, and reducing the association between consecutive packet delivery events all improve with increasing *path diversity*. Path diversity is the use of many paths to deliver packets from the source to the sink. Spreading consecutive packets through different routes on the network depends on the scheduling of links in the network and the routing topology.

Other common objectives for scheduling algorithms are to increase the *throughput* of the network and to ensure *fairness* for different sessions in the network. We will not be focusing on these objectives in our scheduling algorithm.

Aside from the general scheduling objectives mentioned above, we also have *operational objectives* (or constraints) which are concerned with the implementation of the distributed scheduling algorithm. These are to

- reduce the *communication* overhead for computing a schedule, and
- reduce the *memory* and *processing* on each node necessary to compute and operate the schedule.

The communication overhead is measured in packets. Since we assume only one packet can be sent per node in each time slot, we can also measure the communication overhead by the number of rounds of coordination needed to compute the schedule, where one round

corresponds to a fixed number of time slots. The memory usage of the scheduling algorithm depends both on the algorithm's space complexity and whether we need to save the computed schedule for use later.

3.2.3 Scheduling Objectives on Basic Schedules

In the following subsections, we will evaluate the packet latency, the packet path diversity, and the communication and computational complexity of *repeating schedules*, *Greedy Maximal Matching (GMM) schedules*, and modified Greedy Maximal Matching (GMM2) schedules.[2] These schedules are for Unicast Path Diversity (UPD) and not for flooding algorithms since they only have unicast link transmissions. All of our simulation examples will be on the width-3 path topology. Unless specified otherwise, all the links have probabilities equal to 0.9. Link success or failure during a time slot is independent of whether the link succeeded or failed in previous time slots. This matches the link failure model of the flow metric in Section 2.3.1.

A repeating schedule consists of a *superframe* that is repeated over time. A superframe is a block of consecutive *scheduled time slots*, where each scheduled time slot specifies the links in the network that are simultaneously transmitting during the time slot. In this dissertation, we will refer to any schedule that does not explicitly repeat superframes as a *non-repeating schedule*, including schedules that may "unintentionally" generate a scheduled time slot that was used earlier in the schedule. The examples in the following subsections use the short repeating schedule depicted in Figure 3.5.

A greedy maximal matching schedule is a non-repeating schedule generated by a randomized *Greedy Maximal Matching* (GMM) algorithm.[3] The algorithm randomly selects a link (u, v) from the connectivity graph G to add to the matching and removes all links that are incident on u or v to get the graph G'. The algorithm repeats this on the new graph G' until there are no more links in the remaining graph. All the links that have been selected

[2]The author does not claim to have invented these scheduling algorithms, even though they were not presented in the literature review in Section 3.1.

[3]Name clarification: Note that [63] calls the *Greedy Maximal Weighted Matching* (GMWM) algorithm the Greedy Maximal Matching algorithm. There, links with a larger weight are selected first. The GMM algorithm is the same as the GMWM algorithm when all links have equal weights.

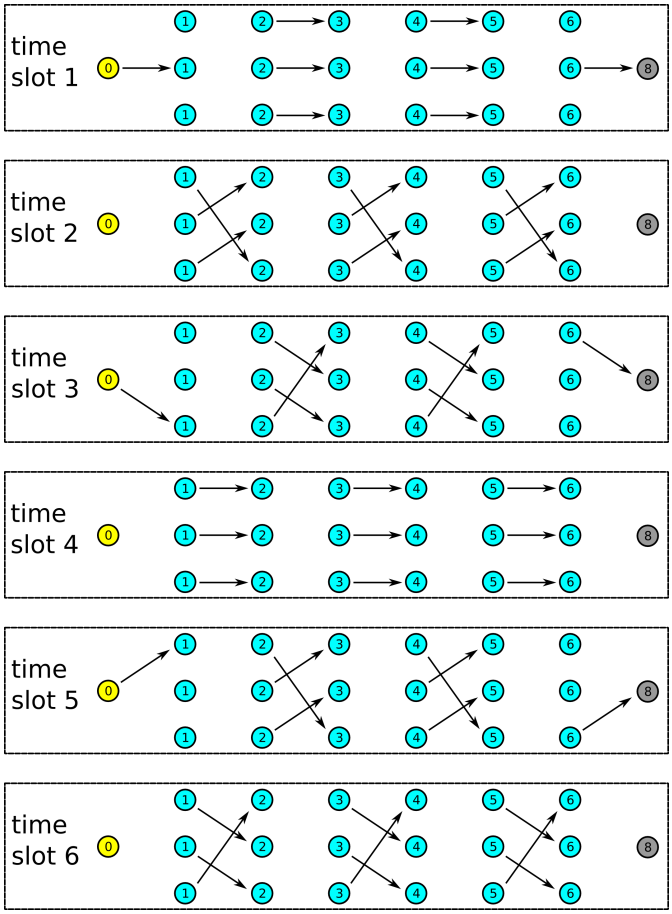

Figure 3.5. UPD repeating schedule for routing on a width-3 path topology used in Examples 3.2.1 and 3.2.2 to generate Figures 3.7 and 3.8. This is the same UPD schedule depicted in Figure 4.13.

are part of the maximal matching for the current time slot, and the algorithm is repeated for the next time slot. On the other hand, a *GMM repeating schedule* is a repeating schedule which uses the GMM algorithm to generate the scheduled time slots for its superframe. If we use GMM scheduling to generate an infinite length non-repeating schedule, each node in the network will be scheduled an equal number of times to transmit packets on each of its outgoing links. This matches the assumption of the flow metric routing model.

The GMM algorithm is not suited for generating a short block of time slots for repeating over time because it may not schedule all the links in the topology. In fact, the GMM schedule generated for our examples in the following subsections took 23 time slots before it scheduled each link at least once. We modified the GMM algorithm to keep track of the set of links $\tilde{\mathcal{E}}$ that have not been selected by previous time slots so that the links from $\tilde{\mathcal{E}}$ can

be selected to transmit in the current time slot before the other links in $\mathcal{E}\backslash\tilde{\mathcal{E}}$. When $\tilde{\mathcal{E}} = \emptyset$, we reset $\tilde{\mathcal{E}}$ to $\mathcal{E}$. We call this modified greedy maximal matching algorithm the *GMM2 algorithm*. The GMM2 schedule generated for our examples in the following subsections took only 7 time slots to schedule each link at least once.

Both the GMM and GMM2 schedules use randomization to generate scheduled time slots. Therefore, they are also referred to as *randomized schedules*. On the other hand, the short repeating schedule in Figure 3.5 was manually created for our topology to schedule all links in the network at least once in the superframe.

All the simulations in the following examples have 1000 runs, where each run simulates the end-to-end delivery of one packet with no queuing of other packets in the network. The source generates packets on every odd time slot. For the repeating schedule, these are the time slots when the source is scheduled to transmit to a downstream node.

Latency

Scheduling in TDMA networks has a large impact on the delay of packets through the network. The two largest sources of end-to-end delay are *transmission delay* and *queuing delay*. We define the transmission delay as the delay from the number of hops a packet takes to reach the sink (routing path), the wait at a node for the next outgoing transmission (schedule), and the number of link retransmissions while routing the packet. We define the queuing delay as the delay at a node while waiting for other packets queued at the node to be transmitted first before the packet can be transmitted. Even in networks where the sources generate packets at a fixed rate below the capacity of the network, queuing can still occur because of link retransmissions, packets traveling along different length paths, and the sharing of relay nodes between multiple sessions.

Transmission delay can be reduced by selecting schedules which reduce the waiting time between transmitting on consecutive hops, as shown in the simple example of Figure 3.6. Note that the transmission delay is actually highly dependent on the link probabilities. The tools presented in Chapter 4 can be used to show that the average transmission delay may

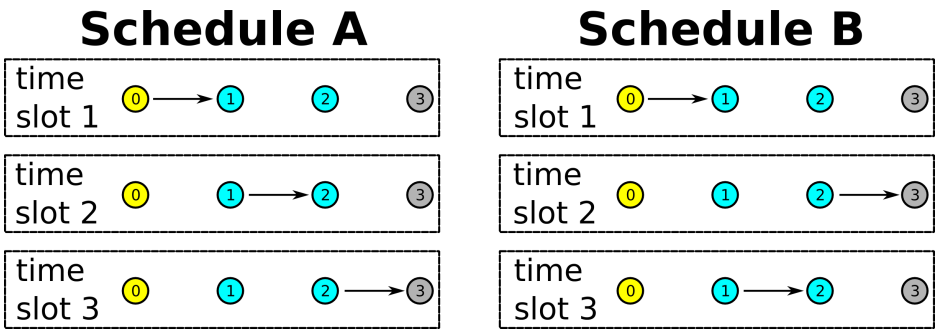

Figure 3.6. Transmission delay depends on the schedule. The transmitting links at each time slot are indicated, and the group of time slots (a superframe) are repeated over time. Schedule A has a lower transmission delay than Schedule B because nodes 1 and 2 need to wait longer between receiving a packet and sending the packet.

be lower if you schedule a poor link to transmit multiple times on consecutive time slots than if you schedule the poor link to transmit only once, assuming each link transmission is independent of past link transmissions. For instance, in Schedule A of Figure 3.6 if all the links had probability 0.1, a schedule repeating the block of time slots $(1, 1, 2, 2, 3, 3)$ instead of $(1, 2, 3)$ would yield a lower transmission delay.

Example 3.2.1 (Packet latency for basic schedules). Figure 3.7 shows the packet latency from simulations under the three different basic scheduling schemes. Our repeating schedule was designed to reduce latency in networks with high link probabilities by scheduling the nodes in the even and odd columns to transmit on alternate time slots. The repeating schedule has the lowest expected packet latency, while the GMM schedule has the highest expected packet latency.

□

Example 3.2.1 clearly shows that careful scheduling can significantly reduce the latency in a network. The algorithm described in Section 3.3 will use the concept of "alternating transmissions from even and odd columns" to generate schedules with low packet latency.

Spreading Packet Paths

We would like a schedule that spreads packets evenly over the different paths in the network in order to reduce the chance of queuing packets in the network, increase reliability,

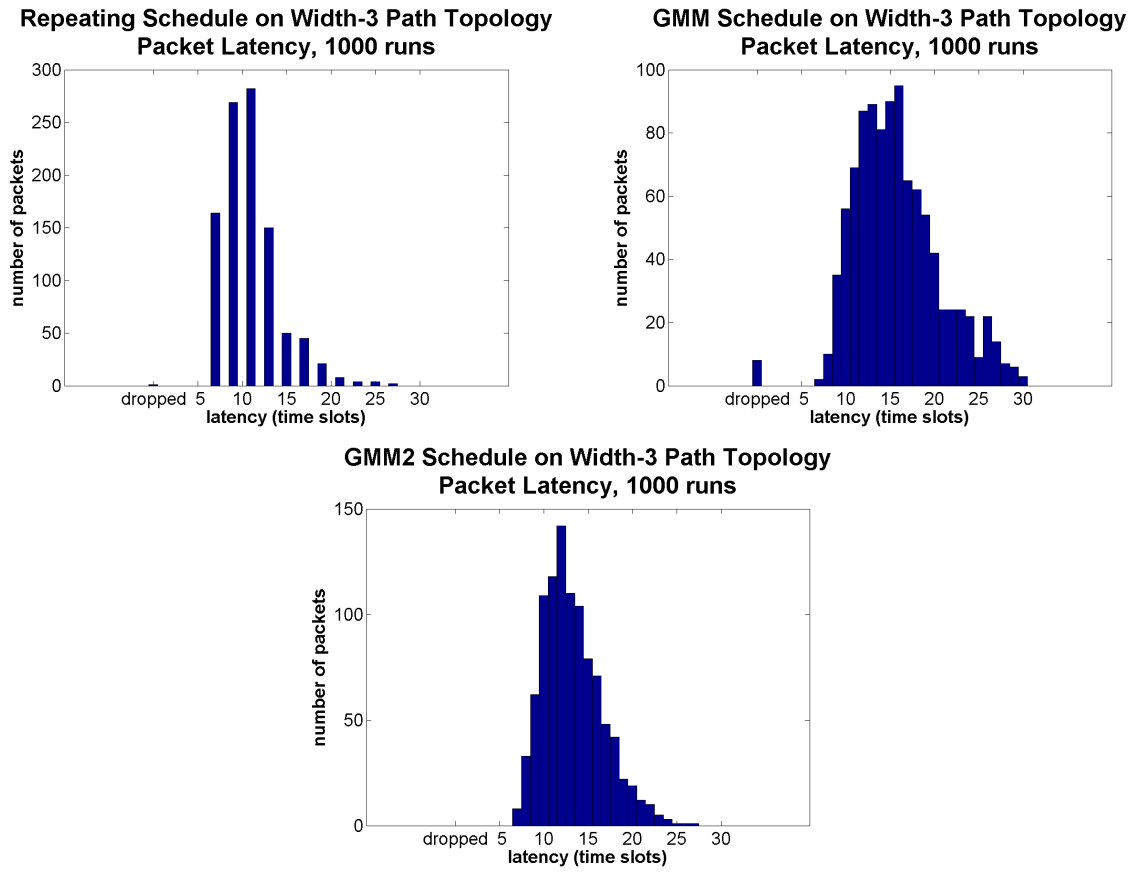

Figure 3.7. Packet delay on a width-3 path topology from simulations using the repeating schedule depicted in Figure 3.5 and the GMM and GMM2 schedules. Packets with a delay over 30 time slots were dropped.

and reduce the association of delivery events. As mentioned in Section 2.3, many packets may be trapped at a node that is shared by many paths if the all the node's outgoing links fail. Also, utilizing fewer paths may lead to more queuing of packets in the network because some links will be used close to their capacity (nodes receive packets very close to the rate that they can relay them).

Short, repeating schedules often do not spread packets throughout the different paths in the network evenly. The next example shows that many packets may converge on certain relay nodes in a network under short, repeating schedules.

Example 3.2.2 (Packet paths for short repeating schedules). Simulations of end-to-end packet delivery for a width-3 path topology using the short repeating schedule of Figure 3.5 show that packets may not utilize all the links in the network equally. In fact, packets generated at the source on the same time slot of the superframe tend to follow similar paths. Since there are six time slots in the superframe of the repeating schedule, the source generates a packet on the same time slot of the superframe on every third simulation run. For example, the source generates packets on time slot 1 of the superframe for runs 1, 4, and 7.

The left of Figure 3.8 shows the fraction of packets that traverse each link from all the simulation runs where a packet was generated for transmission on the first time slot of the superframe. Notice that there is a clear preference for a particular path through the network. This was also observed on the simulation runs where the packet was generated at the third and fifth time slots (not shown), mostly because we have high link probabilities for all the links in this routing topology. In fact, if the link probabilities were equal to 1, there would be one preferred path through the network for packets generated in the same time slot of the superframe. The right of the figure shows the fraction of packets that traverse a particular link from all 1000 simulation runs. Here, also, some links are underutilized while others carry the bulk of the traffic.

□

One approach to get more even path utilization in Example 3.2.2 is to carefully design

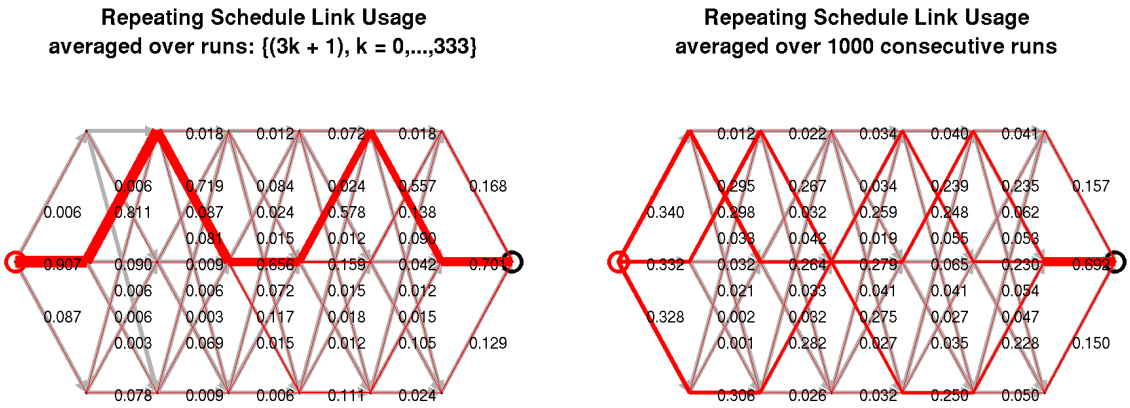

Figure 3.8. Link usage on a width-3 path topology from a simulation using the schedule from Figure 3.5. The source is circled in red, the sink is circled in black, and each link is labeled with the fraction of packets that traversed the link. The red lines overlaid on the links have widths that are directly proportional to the fraction of packets that traverse that link. See text for details.

more scheduled time slots to add to the superframe. However, we would need fairly long superframes to get even path utilization, particularly in networks with high link probabilities. A trivial scenario to show this is a routing topology where all the links in the network have probability equal to 1. In this topology, the runs where the source generates a packet on the same superframe time slot would all send packets down one path. The length of the superframe would need to grow linearly with the number of unique paths we wish the packets to traverse through the network.

The next example demonstrates that we can get even path utilization with longer schedules, even without carefully scheduled time slots (i.e., random, greedily generated schedule time slots).

Example 3.2.3 (Packet paths for randomized schedules). This example examines the performance of non-repeating, randomized schedules. We use the randomized Greedy Maximal Matching (GMM) algorithm to choose the set of transmitting links in each time slot. Over a long time horizon, each node in the network is scheduled to transmit packets an equal number of times on each of its outgoing links. This matches the assumption of the flow metric routing model.

The left of Figure 3.9 shows that the path utilization of the network is fairly even after

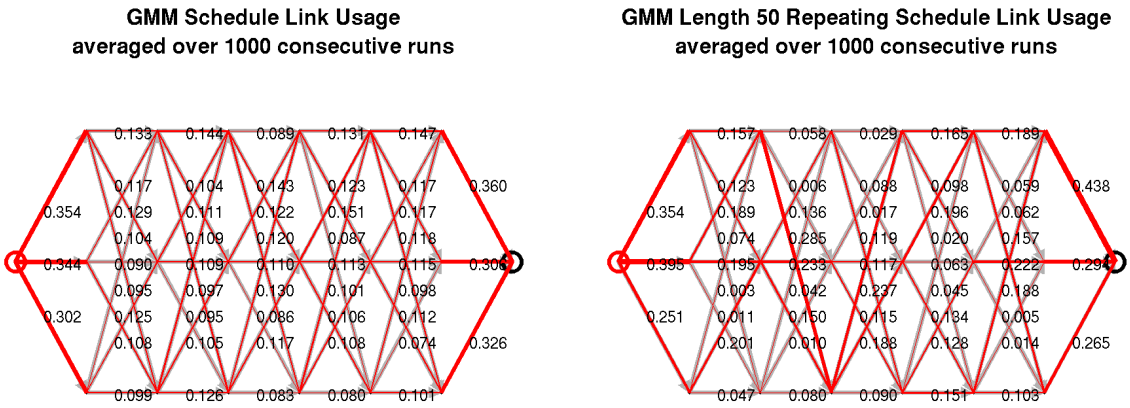

Figure 3.9. Link usage on a width-3 path topology from a simulation using GMM schedules. The source is circled in red, the sink is circled in black, and each link is labeled with the fraction of packets that traversed the link. The red lines overlaid on the links have widths that are directly proportional to the fraction of packets that traverse that link. See text for details.

using the GMM schedule over a long time horizon. However, if we were to use the GMM algorithm to generate 50 time slots for a superframe in a repeating schedule, we would again get fairly uneven path utilization in the network, as shown on the right of Figure 3.9. The path utilization is more even if we use a longer block of time slots from our randomly generated schedule for repetition. For comparison, there are $3^6 = 729$ unique paths through this network.

Figure 3.10 shows the path utilization from the GMM2 algorithm. As with the GMM schedule, the path utilization of the GMM2 schedule after a long time horizon is fairly even, but if we generate 50 time slots using the GMM2 schedule and repeat it, the path utilization is noticeably less even.

□

Cluster-Based Scheduling

As mentioned at the beginning of the chapter, there are tradeoffs between using centralized and distributed scheduling. Centralized scheduling requires collecting information about the routing topology from the network and distributing the computed schedule back

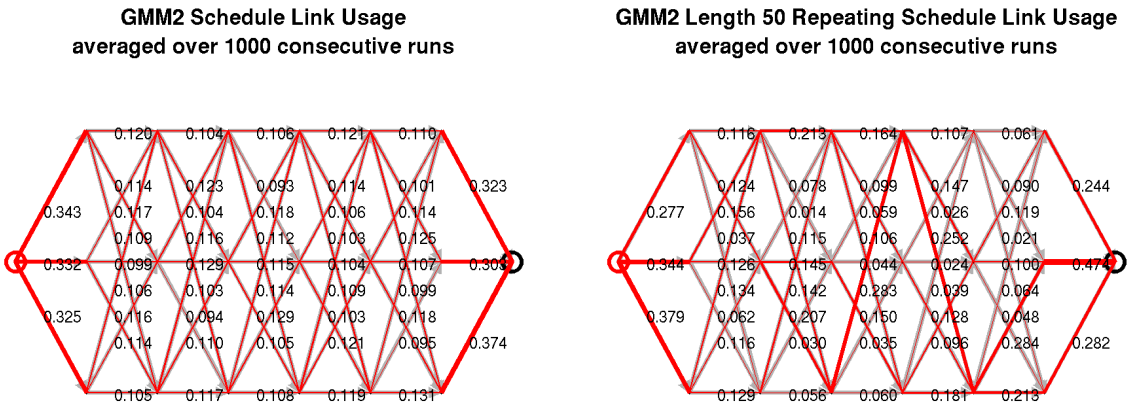

Figure 3.10. Link usage on a width-3 path topology from a simulation using GMM2 schedules. The source is circled in red, the sink is circled in black, and each link is labeled with the fraction of packets that traversed the link. The red lines overlaid on the links have widths that are directly proportional to the fraction of packets that traverse that link. See text for details.

to the nodes, which can have significant bandwidth overhead. Distributed scheduling may be suboptimal in meeting our scheduling objectives because it relies on local (not global) information, and may take several rounds of communication to compute.

Both the GMM and GMM2 algorithms presented earlier are centralized, particularly GMM2 which needs to keep track of links that have been scheduled in previous time slots from all parts of the network. GMM may be implemented by randomized, distributed algorithms similar to the MIS algorithm presented in [66], but there is significant communication overhead to generate the matchings for each time slot.

The communication overhead for both centralized and distributed scheduling can be reduced by using repeating schedules. The superframes can be computed using any distributed or centralized scheduling algorithm (GMM, GMM2, or others) and stored on the nodes so that the network only needs to compute it once. However, the length of the superframe may need to be fairly large in some networks to have even path utilization, as was shown in Example 3.2.3. It may be difficult to store long schedules on resource constrained nodes (and for centralized algorithms, difficult to disseminate the schedule throughout the network).

An alternative is to try to generate the time slots for the schedule on the nodes in the

network "just-in-time" for use rather than precomputing and storing the entire schedule. The challenge lies in devising algorithms to compute these schedules with minimal or no communication between the nodes, except possibly during an initial set up period. Instead of precomputing schedules, the network will form local clusters that will exchange information in the initial set up period and then generate local schedules with minimal communication thereafter.

Cluster-based scheduling takes advantage of the local nature of sub-optimal scheduling. By grouping nodes into clusters that share local routing topology information, the global scheduling problem is broken into smaller, tractable subproblems that can be solved in parallel. Furthermore, each of the nodes in the cluster can run the exact same scheduling algorithm in parallel, meaning that they will not need to exchange any information after the initial information exchange (except to occasionally synchronize when the routing topology has changed). We say *a cluster is scheduled* in a time slot if we use the cluster to constrain the scheduling problem in the time slot such that links between the nodes in the cluster are scheduled. The set of clusters to be scheduled in each time slot form a *partition* of the network, where every node belongs to a cluster and each cluster is disjoint. Formally, a partition Ξ is a set of clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_M\}$, where each cluster $\mathcal{C}$ is a subset of $\mathcal{V}$ and the clusters in a partition satisfy the properties $\forall m, n \in \{1, \ldots, M\}, \quad \mathcal{C}_m \cap \mathcal{C}_n = \emptyset$ and $\bigcup_m \mathcal{C}_m = \mathcal{V}$. *A partition is scheduled* in a time slot if the clusters in the partition are scheduled in the time slot. The partitions for different time slots may be different, meaning that a node may belong to different clusters on different time slots. If we only had one partition of the network for all time slots, none of the links *between* the clusters would ever be scheduled.

The performance of a cluster-based scheduling algorithm depends heavily on the criteria used to form clusters. Larger clusters allow for better scheduling of links within the cluster, but are also harder to coordinate (more state, more information exchanged between nodes). Nodes are formed into clusters based on whether they share links and whether they have overlapping interference regions. For instance, we can select a *leader node* and form a cluster

with all its 2-hop neighbors. The hop count assignments used to form the routing topology may also be employed to help group nodes into clusters, as we will see in Section 3.3.

3.3 Alternating Partition Local Matching

This section describes the *Alternating Partition Local Matching* (APLM) algorithm, which was designed with the scheduling objectives of the previous section in mind. APLM operates using two primary mechanisms: alternate partitions are selected for scheduling over time and, given the partition for a time slot, a pseudorandom generator is used to coordinate and generate randomized matchings within the clusters to be scheduled. As we shall see in Section 3.3.1, APLM was designed to work well for a class of routing topologies called *layer-to-layer* topologies.

The scheduling algorithm works in two phases. In the *initial setup phase* the algorithm forms multiple partitions on the network and the nodes within each cluster of each partition exchange information. In the *schedule generation phase* a partition is selected for scheduling the next time slot and the scheduled time slot is generated without communication between the nodes. The schedule generation phase runs simultaneously with the delivery of data through the network. Notice that the initial setup phase requires loose global coordination between the nodes while the the schedule generation phase does not. The input to the scheduling algorithm is the routing topology.

The 2-hop neighbor clustering algorithm for the initial setup phase of APLM is given in Algorithm 8, written as a centralized algorithm for ease of reading. It assumes that the routing topology assigns each node a hop count (e.g., minimum hop count or robust hop count). The hop count is used as a partial ordering of the nodes to help in the construction of the partitions and the construction of clusters within each partition. The inputs are the graph $G = (\mathcal{V}, \mathcal{E})$ and the integer hop count vector $\boldsymbol{\hbar} = \lfloor \boldsymbol{\hbar}^f \rfloor$, which are obtained from the routing topology. The output is an ordered sequence of partitions of the network (Ξ_1, Ξ_2) (in general, there could be more partitions). Notice that the random leader election scheme to help construct each cluster can be implemented in a distributed manner similar to the

method used by Luby to get an MIS [66]. We constrain nodes within a cluster to be within a 2-hop communication range in addition to the constraint that nodes within a cluster must have integer hop counts that differ by 1 or 0 because the latter constraint alone does not guarantee that nodes within a cluster are within close communication range of each other. For instance, there are nodes which are separated by many hops but have the same hop count in a topology with a sink in the middle of the network and two paths branching out in opposite directions.

Algorithm 8 APLM_Clustering (Initial Setup Phase)

Input: $G = (\mathcal{V}, \mathcal{E}), \boldsymbol{\hbar}$

Output: (Ξ_1, Ξ_2)

$\Xi_1 := \emptyset, \Xi_2 := \emptyset$

for $k := 1$ to 2 **do**

  for $h := k$ to $\max_i \hbar_i$, h incrementing by 2 **do**

    $\mathcal{H} := \{i \in \mathcal{V} : \hbar_i \in \{h, h-1\}\}$

    while $\mathcal{H} \neq \emptyset$ **do**

      Randomly select a leader node $\ell \in \mathcal{H}$.

      Let $\mathcal{N}_\ell^{\mathcal{H},2}$ be the set of 2-hop neighbors of ℓ on the subgraph induced by $\mathcal{H}$.

      $\Xi_k := \Xi_k \cup \{\mathcal{N}_\ell^{\mathcal{H},2}\}$

      $\mathcal{H} := \mathcal{H} \backslash \mathcal{N}_\ell^{\mathcal{H},2}$

    end while

  end for

end for

After forming clusters, the nodes in each cluster exchange a list of their links and each cluster leader node selects a pseudorandom seed $\boldsymbol{s}_\mathcal{C}$ and disseminates it to all the nodes in the cluster. This occurs for both clusters in Ξ_1 and clusters in Ξ_2, so each node will have two sets of cluster information.

The schedule generation phase of APLM is given in Algorithm 9, again written as a centralized algorithm for ease of reading. It assumes that all nodes in a cluster $\mathcal{C}$ know the other nodes in the cluster, the links between nodes in the cluster, and a shared seed

$s_{\mathcal{C}}$ for a pseudorandom number generator. Each node runs the same schedule generation algorithm on the same input, so they agree on the scheduling of the cluster for the current time slot without any communication. The pseudorandom generator is used to "randomize" the matchings generated in each time slot to get better path utilization, and uses the same seed across the nodes in $\mathcal{C}$ so the nodes compute the same schedule. The output of the algorithm, $\boldsymbol{\mathcal{M}}$, is the sequence of matchings representing the scheduled links at each time slot.

Most of the complexity in APLM happens in the initial setup phase, while forming partitions of the network. We have imposed the restriction that each cluster contain only nodes whose hop counts differ by 1 or 0, which limits the size of clusters. When a node selects itself as a leader, it makes a 2-hop broadcast to those nodes that have not yet joined a cluster. These nodes respond with their 1-hop neighbor list (representing all the links used by the node). The leader collects all these 1-hop neighbor lists and makes a 2-hop broadcast to all the nodes in the cluster sharing all the links between the nodes in the cluster and a pseudorandom seed for the schedule generation phase. Thus, each node keeps track of all the nodes and links in the cluster as well as the random seed, which take $O(|\mathcal{V}' + \mathcal{E}'|)$ space if there are $\mathcal{V}'$ nodes in the cluster and $\mathcal{E}'$ links between them.

The schedule generation phase uses greedy maximal matching without replacement (GMM2) on the subgraph induced by the nodes in a cluster. A cluster with $\mathcal{E}'$ edges between nodes in the subgraph takes $O(|\mathcal{E}'|)$ time and $O(|\mathcal{E}'|)$ space to generate the local greedy maximal matching.

3.3.1 Examples and Discussion

Our first example shows that APLM works well on layer-to-layer topologies. A layer-to-layer topology is a topology where nodes with a hop count $\hbar$ only have outgoing links to nodes with a hop count $\hbar - 1$ and incoming links from nodes with a hop count $\hbar + 1$. A "layer" is a set of nodes that share the same hop count. The width-3 path topology is an example of a layer-to-layer topology.

Algorithm 9 APLM_Sched_Gen (Schedule Generation Phase, uses GMM2)

Input: $G = (\mathcal{V}, \mathcal{E}), (\Xi_1, \Xi_2), \boldsymbol{s}$ ▷ $\boldsymbol{s}$ is a vector of pseudorandom seeds.

Output: $\boldsymbol{\mathcal{M}} = \{\mathcal{M}_t\}_{t=1}^{\infty}$ ▷ $\mathcal{M}_t$ is the set of active links at time slot t.

for $k := 1$ to 2 **do** ▷ Initialization.

  for all $\mathcal{C} \in \Xi_k$ **do**

    $\mathcal{E}_{\mathcal{C}}^* := \{(u, v) \ : \ u, v \in \mathcal{C}\}$ ▷ $\mathcal{E}_{\mathcal{C}}^*$ are edges not scheduled recently.

  end for

end for

for $t := 1$ to ∞ **do**

  $\mathcal{M}_t := \emptyset$

  Let $k = t \mod 2$

  for all $\mathcal{C} \in \Xi_k$ **do** ▷ Run simultaneously on all clusters in Ξ_k.

    [Run the following simultaneously on all nodes in cluster]

    $\mathcal{E}' := \{(u, v) \ : \ u, v \in \mathcal{C}\}$

    while $\mathcal{E}_{\mathcal{C}}^* \cap \mathcal{E}' \neq \emptyset$ **do**

      Select a (pseudo)random $e = (i, j) \in \mathcal{E}_{\mathcal{C}}^* \cap \mathcal{E}'$ using $\boldsymbol{s}_{\mathcal{C}}$.

      $\mathcal{E}_{\mathcal{C}}^* := \mathcal{E}_{\mathcal{C}}^* \backslash \{e\}$

      $\mathcal{E}' := \mathcal{E}' \backslash \{(u, v) \ : \ u = i \text{ or } v = j\}$

      $\mathcal{M}_t := \mathcal{M}_t \cup \{e\}$

    end while

    while $\mathcal{E}' \neq \emptyset$ **do**

      Select a (pseudo)random $e = (i, j) \in \mathcal{E}'$ using $\boldsymbol{s}_{\mathcal{C}}$.

      $\mathcal{E}' := \mathcal{E}' \backslash \{(u, v) \ : \ u = i \text{ or } v = j\}$

      $\mathcal{M}_t := \mathcal{M}_t \cup \{e\}$

    end while

    if $\mathcal{E}_{\mathcal{C}}^* = \emptyset$ **then**

      $\mathcal{E}_{\mathcal{C}}^* := \{(u, v) \ : \ u, v \in \mathcal{C}\}$

    end if

  end for

end for

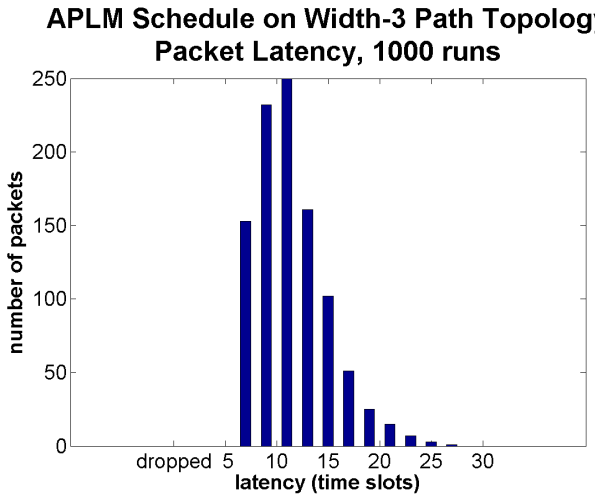

Figure 3.11. Packet delay on a width-3 path topology from simulations using APLM schedules. Packets with a delay over 30 time slots were dropped.

Example 3.3.1 (APLM on a width-3 path topology). This example runs the APLM scheduling algorithm on the same routing topology and simulation setup described for the basic schedules in Section 3.2.3. A comparison of Figure 3.11 with Figure 3.7 suggests that the packet delays from APLM schedules are slightly better than the packet delays from GMM2 schedules, but not as good as the packet delays from the repeating schedule in Figure 3.5. This is expected, as the latter schedule was designed to reduce latency on the topology when the links have high success probabilities.

Similarly, a comparison of Figure 3.12 with Figures 3.9 and 3.8 shows that the APLM schedule has comparable path utilization with the GMM and GMM2 schedules, and better path utilization than a short repeating schedule.

□

Unfortunately, the APLM algorithm in its current form does not work well on some randomly generated topologies.

Example 3.3.2 (APLM on a randomly generated topology). We randomly generated a connectivity graph and constructed a robust routing topology using the setup of Example 2.4.1 (30 nodes in 10×10 grid, $K = 100$, etc.), and then ran the APLM algorithm to generate a schedule. The scheduled time slot depicted on the left of Figure 3.13 shows that because

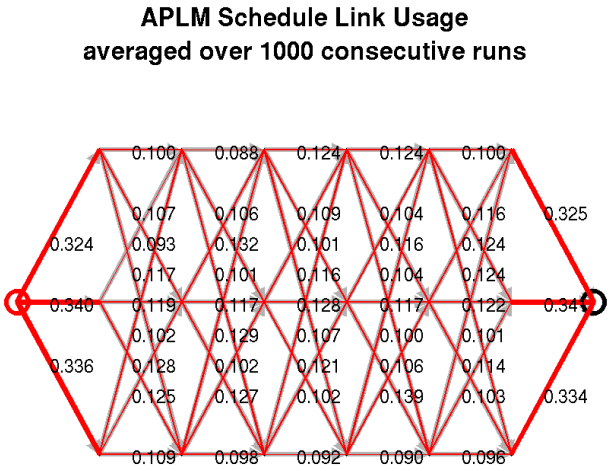

Figure 3.12. Link usage on a width-3 path topology from a simulation using an APLM schedule. The source is circled in red, the sink is circled in black, and each link is labeled with the fraction of packets that traversed the link. The red lines overlaid on the links have widths that are directly proportional to the fraction of packets that traverse that link.

APLM has restrictive clustering rules, it may schedule far less links than a maximal matching. For reference, the clusters in the partition used to schedule the time slot is depicted on the right of Figure 3.13, where nodes in the same layer are assigned the same color. Many links are not scheduled because they are between nodes with hop counts that differ by more than 1.

Figure 3.14 compares the packet delay from using APLM and GMM2 scheduling on the same random topology used in Figure 3.13. APLM drops significantly more packets because it tends to have a higher packet delay (packets with a delay over 30 are considered to have been dropped). Figure 3.15 compares the path utilization from using APLM and GMM2 on this random topology. Notice that there are links used by the GMM2 schedule which are not used by the APLM schedule. More research needs to be done on how to improve APLM's path utilization.

□

APLM, as described in Section 3.3, needs a better clustering mechanism in the initial setup phase to get better performance on random topologies. Certainly, using larger clusters (instead of restricting clusters to adjacent layers) can result in schedules with better path

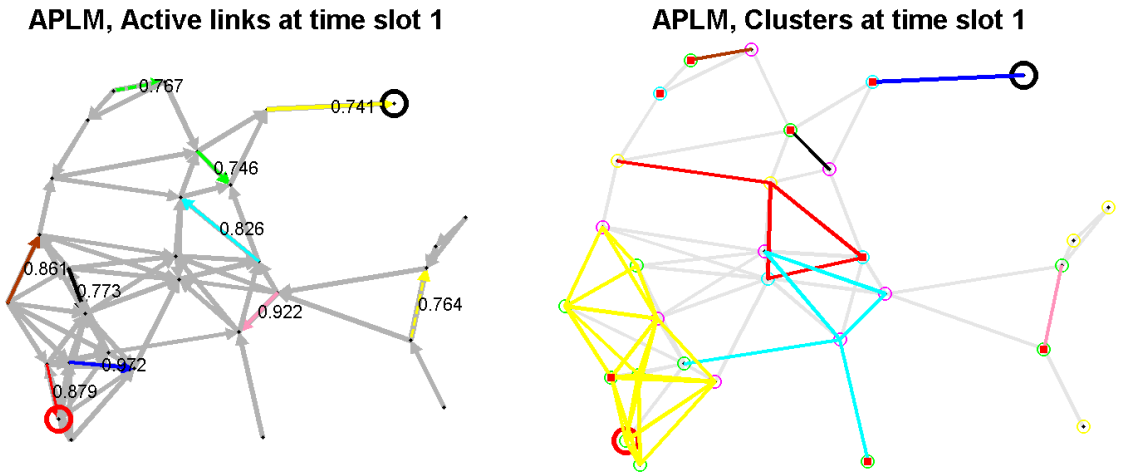

Figure 3.13. On random topologies APLM does not schedule many links in a time slot, far fewer links than a maximal matching. (left) The active links in a sample time slot from an APLM schedule, labeled by their link probabilities. (right) The clusters used to schedule the links in the time slot. Nodes in the same layer are circled in the same color and cluster leader nodes are marked by a red square. The links within a cluster are drawn in the same color (blue, black, or red), to differentiate them from the links in nearby clusters.

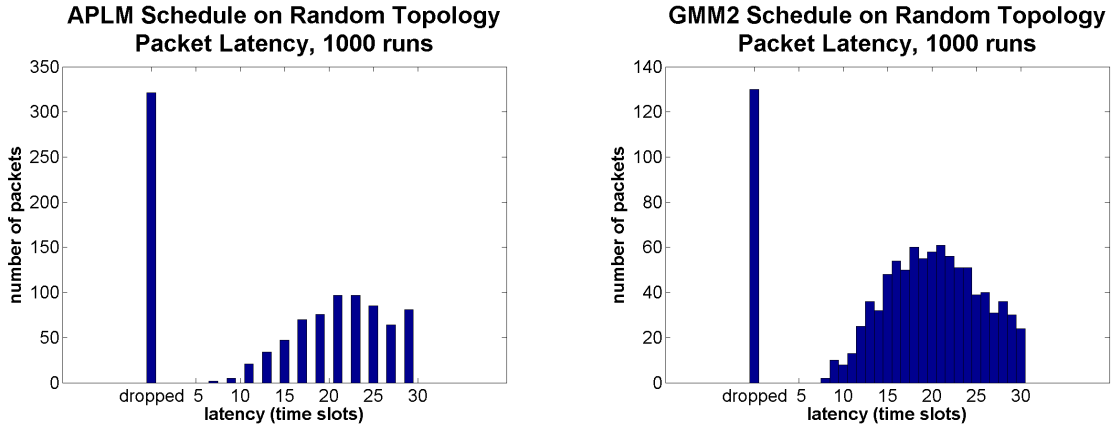

Figure 3.14. Packet delay on the same random topology used in Figure 3.13 from simulations using APLM and GMM2 schedules. Packets with a delay over 30 time slots were dropped.

utilization and lower packet delays because of better local matchings, at the expense of more state and more coordination between nodes. Directions for further exploration will be presented in Chapter 5. The evaluation of APLM on the width-3 path topology shows that it may be useful in wireless networks with layer-to-layer routing topologies.

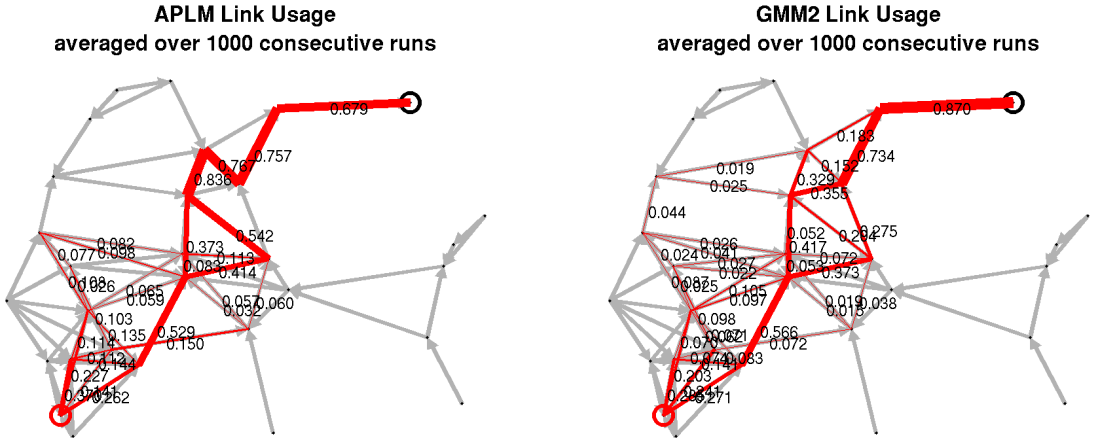

Figure 3.15. Link usage on the same random topology used in Figure 3.13 from simulations using APLM and GMM2 schedules. The source is circled in red, the sink is circled in black, and each link is labeled with the fraction of packets that traversed the link. The red lines overlaid on the links have widths that are directly proportional to the fraction of packets that traverse that link.

Chapter 4

Metrics for Real-Time Wireless Networked Systems

This chapter is concerned with modeling the end-to-end delivery of packets through a TDMA wireless mesh network. The primary objective of the models developed in this chapter is to compute the *end-to-end connectivity* of the network as a function of latency, $p_{\text{net}}^{(t_d)} \in [0, 1]$. The end-to-end connectivity is the probability of receiving a packet t_d time units after it was sent, and serves as an *abstraction* of the network that is presented to the rest of the wireless networked system.

For instance, the end-to-end connectivity metric $p_{\text{net}}^{(t_d)}$ can be used to help design controllers or prove stability properties of *Networked Control Systems* (NCSs). Section 4.1 gives some background on the theory of Networked Control Systems.[1] The end-to-end connectivity metric can also be used to check the feasibility of running a wireless networked system over the network. If the network is under-provisioned for the task, the system designer may choose to add more nodes to the wireless network or use different networking algorithms that better match the requirements of the wireless networked system. Section 4.5 gives examples of how to use the connectivity metric for system feasibility analysis.

[1] This section stands alone from the rest of the dissertation. Because we are running out of good unused symbols, this section sometimes reuses math notation in a manner inconsistent with the rest of the dissertation.

We propose Markov chain models for two classes of networking protocols, Unicast Path Diversity (UPD) and Directed Staged Flooding (DSF). These models are constructed given the routing topology, link probabilities, and schedule on the network. They model the end-to-end delivery of one packet for one session, and do not model queuing in the network. Furthermore, they do not model the association / correlation of consecutive packet delivery events. The models assume the topology and schedule are static for the duration of packet delivery, and therefore do not model network maintenance mechanisms common to ad-hoc networking protocols such as route discovery, routing loop detection, beaconing for time synchronization, etc.

In addition to computing the end-to-end connectivity metric, the models will be used to find the sensitivity of the calculations to link probability estimation error, identify *hot spots* in the network where traffic is concentrated, and estimate the radio energy consumption on the nodes from relaying packets.

4.1 Background on Networked Control Systems

The theory of *Networked Control Systems* (NCS) can be used to aid the design and performance evaluation of wireless networked systems. A networked control system is defined to be a spatially distributed system of controllers, sensors, and actuators that share a band-limited digital communication network [43] (See Figure 4.1). The study of NCSs focuses on the impact a shared, lossy communication network with variable delay has on the performance of the closed loop control system. The literature in this area has largely focused on packet networks, especially since many wired SCADA and Distributed Control systems are migrating to TCP/IP networks for cost and interoperability [74].

This section will highlight some important contributions to the theory of networked control systems to give a sense of how network conditions translate to system performance. Although there often can be multiple interacting control loops in a networked control system, for clarity this exposition will focus on single-loop NCSs (See Figure 4.2). A more comprehensive overview of NCSs can be found in [43].

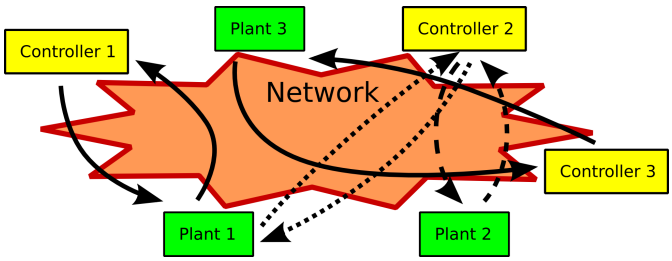

Figure 4.1. The components of a Networked Control System (NCS). There can be many control loops running over the same network, and multiple controllers per plant or multiple plants per controller.

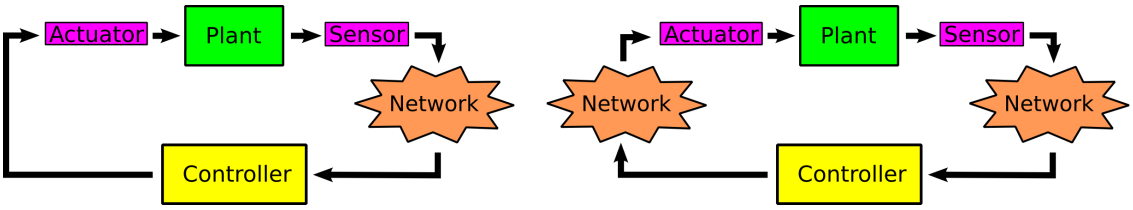

Figure 4.2. (left) One channel, single-loop NCS. Note that although the network is depicted between the sensor and the controller, it can also be between the actuator and the controller. This latter situation is less common in practice. (right) Two channel, single-loop NCS. (Figure after [43].)

The current literature on networked control systems focuses on system stability / performance analysis and controller synthesis, where the network introduces *packet drops*, *variable sampling*, *variable delay*, and *measurement / control quantization error*. The literature does not cover these areas equally — for instance, there is a heavier emphasis on studying system stability under packet drops and delay — but NCSs is still an active area of research. Also, much of the current literature analyzes control of a simple plant, such as a discrete-time LTI (linear time-invariant) plant with Gaussian noise and disturbance, to focus on the control issues introduced by using a network.

There are two types of system stability and performance guarantees: *deterministic* and *stochastic*. Deterministic system guarantees often rely on absolute guarantees on network performance — for instance, packets in the network have a bounded delay, or that no more than a fixed number of consecutive packets can be dropped. These network guarantees may be impossible to enforce on a wireless network. The alternative is to use a stochastic characterization of the network to provide stochastic system guarantees. For instance, if $\boldsymbol{x}_k$ is a random process representing the state of the system at time k, one can say the system is

- *almost surely stable* if

  $\mathbb{P}(\sup_{k\in\mathbb{N}}\|\boldsymbol{x}_k\| < \infty) = 1$

- *stable in the 2m-th moment* if

  $\sup_{k\in\mathbb{N}} \mathbb{E}[\|\boldsymbol{x}_k\|^{2m}] < \infty$

- *asymptotically stable in the 2m-th moment* if

  $\lim_{k\to\infty} \mathbb{E}[\|\boldsymbol{x}_k\|^{2m}] = 0$

- *exponentially stable in the 2m-th moment* if

  $\exists \alpha, \beta > 0$ s.t. $\mathbb{E}[\|\boldsymbol{x}_k\|^{2m}] < \alpha\, \mathbb{E}[\|\boldsymbol{x}_0\|^{2m}] e^{-\beta k}, \forall k \in \mathbb{N}.$

If $m = 1$, one can also say the system is (exponentially / asymptotically) *mean-square* stable. Thus, system guarantees can be made for average performance or the probability of system failure, akin to the usage of "outage probability" (e.g., 99.999% uptime) to characterize telephone communication networks.

The synthesis of controllers for networked control systems that can meet system stability and performance requirements is a hard problem. For tractability, many approaches often restrict the controllers to a certain form — for instance memoryless controllers, linear controllers, or controllers that switch between static control gains. Using simple controllers such as those switching between static control gains may not yield the best system performance, but it can reduce the complexity and computation required to implement the controllers.

The current literature on NCSs often use a point-to-point channel model as an abstraction for the network to study the impact of channel characteristics (packet drops, variable sampling, variable delay, and measurement / control quantization error) on the closed loop system. The way we quantify / measure these channel characteristics, the problem of computing *network metrics*, is studied later in this chapter.

The architecture of a networked control system depends on whether it is a single channel network or a two channel network, and where the network is placed relative the the controller, sensors, actuators, and plant (See Figure 4.2). For instance, if the system has *smart*

sensors that include some computational capabilities with the sensor [115], the sensor can transmit a state estimate over the network instead of a raw measurement. This effectively connects part of the controller — the estimator — to the sensor directly without a network, and results in better system performance [90].

The following two examples consider two networked control systems with stochastic packet drops but no variable delay, no variable sampling, and no quantization error on the transmitted data. Although delays are common in networks and are important for control, these simplified systems match a network architecture where packets arriving after a deadline are dropped. The techniques for dealing with time-delayed systems modeled as delayed differential equations (DDE) are covered in the references of [43]. The examples below follow the notation and exposition from [43].

4.1.1 Estimator Stability under Bernoulli Packet Drops

In [89], Schenato et al. give stochastic guarantees on the stability of optimal estimators for a discrete-time LTI system connected by two-channel feedback where packet drops are modeled as a Bernoulli process (Figure 4.2, right). Particularly, there is a threshold probability p_c such that if the probability of dropping a packet $p > p_c$, then the estimation error covariance of the optimal estimator is unbounded. Furthermore, they show that if the controller receives an acknowledgment of whether the previous control packet reached the actuator, what they term "TCP-like protocols", then the separation principle holds. This means that the optimal estimator and the optimal controller can be designed separately. In fact, the optimal controller is a linear function of the state. On the other hand, if the network does not have acknowledgments, what they term "UDP-like protocols", then the separation principle fails and the optimal controller is in general nonlinear. In [90], Schenato extends this problem to considering estimators that can buffer past measurements to account for out of order packet delivery.

For simplicity, consider just the estimation problem for a discrete-time LTI plant driven

by white Gaussian noise:

$$\left.\begin{aligned} \boldsymbol{x}_{k+1} &= A\boldsymbol{x}_k + \boldsymbol{w}_k \\ \boldsymbol{y}_{k+1} &= C\boldsymbol{x}_k + \boldsymbol{v}_k \end{aligned}\right\} \qquad \forall k \in \mathbb{N}, \quad \boldsymbol{x}_k, \boldsymbol{w}_k \in \mathbb{R}^n, \quad \boldsymbol{y}_k, \boldsymbol{v}_k \in \mathbb{R}^p \tag{4.1}$$

The initial state is $\boldsymbol{x}_0 \sim \mathcal{N}(0, \Sigma)$ (i.e., x_0 has a Gaussian distribution with mean 0 and covariance Σ) and the Gaussian white noises $\boldsymbol{w}_k \sim \mathcal{N}(0, R_w), R_w \geq 0$ and $\boldsymbol{v}_k \sim \mathcal{N}(0, R_v), R_v \geq 0$ are independent. Let (C, A) be detectable and (A, R_w) be stabilizable.

The plant and the estimator is connected by an erasure channel, meaning the receiver knows when a packet is corrupted and can discard it. The lossy channel is modeled by a stochastic process $\boldsymbol{\theta}_k \in \{0, 1\}, \forall k \in \mathbb{N}$ which is independent of $\boldsymbol{x}_0, \boldsymbol{w}_k,$ and $\boldsymbol{v}_k$. Here, $\boldsymbol{\theta}_k = 1$ means a measurement packet reached the sink and $\boldsymbol{\theta}_k = 0$ means it did not.

The goal is to compute the optimal estimate of $\boldsymbol{x}_k$ given all the measurements successfully transmitted up to time $j \leq k$, which is

$$\hat{\boldsymbol{x}}_{k|j} = \mathbb{E}[\boldsymbol{x}_k \mid \boldsymbol{y}_l, \forall l \leq j \text{ s.t. } \boldsymbol{\theta}_l = 1]. \tag{4.2}$$

This is computed recursively using the following Time-Varying Kalman Filter (TVKF):

$$\left.\begin{aligned} \hat{\boldsymbol{x}}_{0|-1} &= 0 \\ \hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{\theta}_k F_k(\boldsymbol{y}_k - C\hat{\boldsymbol{x}}_{k|k-1}) \\ \hat{\boldsymbol{x}}_{k+1|k} &= A\hat{\boldsymbol{x}}_{k|k} \end{aligned}\right\} \qquad \forall k \in \mathbb{N} \tag{4.3}$$

where the matrix gain F_k is computed by

$$\left.\begin{aligned} P_0 &= \Sigma \\ F_k &= \boldsymbol{\theta}_k P_k C^{\mathsf{T}}(CP_kC^{\mathsf{T}} + R_v)^{-1} \\ P_{k+1} &= AP_kA^{\mathsf{T}} + R_w - AF_k(CP_kC^{\mathsf{T}} + R_v)F_k^{\mathsf{T}}A^{\mathsf{T}} \end{aligned}\right\} \qquad \forall k \in \mathbb{N}. \tag{4.4}$$

Here, P_k is the estimation error covariance, or simply

$$P_k = \mathbb{E}[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1})^{\mathsf{T}}], \qquad \forall k \in \mathbb{N}. \tag{4.5}$$

The main result of [89] is stated in the following theorem.

Theorem 4.1.1 (From [89]). *Assume that the packet drop process $\boldsymbol{\theta}_k$ is a Bernoulli process with packet drop probability*

$$p \triangleq \mathbb{P}(\boldsymbol{\theta}_k = 0) \in [0, 1), \qquad \forall k \in \mathbb{N}. \tag{4.6}$$

Then, there exists a critical value $p_c \in (0, 1]$ such that:

- *For all $p \geq p_c$, there is exists some $P_0 \geq 0$ for which the sequence $(\mathbb{E}[P_k])_k$ computed from the TVKF (4.3) is unbounded.*
- *For all $p \leq p_c$ and every $P_0 \geq 0$, the sequence $(\mathbb{E}[P_k])_k$ computed from the TVKF (4.3) is uniformly bounded.*

In general, the critical probability p_c cannot be computed explicitly but $\underline{p} \leq p_c \leq \bar{p}$ where

$$\bar{p} = \frac{1}{(\max |\lambda(A)|)^2} \tag{4.7}$$

and

$$\begin{gathered} \underline{p} = \max\{p \geq 0 : \ \Psi_p(Y, Z) > 0, \ 0 \leq Y \leq I \text{ for some } Y, Z\} \quad \text{where} \\ \Psi_p(Y, Z) = \begin{bmatrix} Y & \sqrt{1-p}(YA + ZC) & \sqrt{p}YA \\ \sqrt{1-p}(A^\mathsf{T}Y + C^\mathsf{T}Z^\mathsf{T}) & Y & 0 \\ \sqrt{p}A^\mathsf{T}Y & 0 & Y \end{bmatrix}, \quad \forall Y, Z \in \mathbb{R}^{n \times n}. \end{gathered} \tag{4.8}$$

In the special case where C is invertible, we have $p_c = \bar{p}$. □

4.1.2 H_∞ Controller Synthesis for MJLS

In [92], Seiler and Sengupta study how to synthesize an H_∞ controller for a one-channel feedback NCS with a discrete-time LTI plant (See Figure 4.2, left). They also derive stability conditions for the closed loop system when the channel can be modeled by the Gilbert-Elliot channel model. In the Gilbert-Elliot channel model packet losses are modeled by a two-state Markov chain (See Figure 4.3), unlike the previous example where packet losses are independent. This is a simple approximation of wireless channel fading where consecutive packets are dropped with high probability. As a result of this channel model, the NCS is modeled as a Markov Jump Linear System (MJLS). Seiler and Sengupta formulate a semidefinite programming problem using the MJLS model to synthesize an H_∞ controller

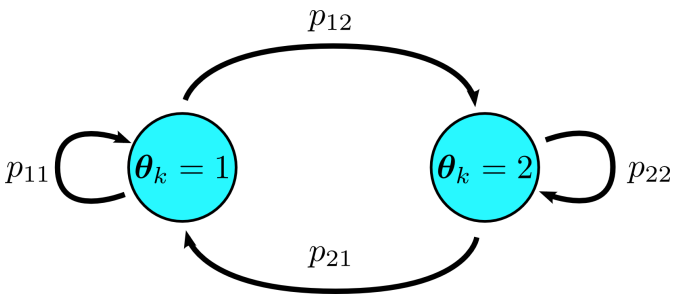

Figure 4.3. The Gilbert-Elliot communication model uses a 2-state Markov chain to model correlated packet losses. Note that if we set $p_{11} = p_{21}$, and $p_{12} = p_{22}$, we get back the Bernoulli packet loss model.

that makes the closed loop system exponentially mean-square stable (but for synthesizing the controller they also assume the Bernoulli packet loss model). We will be focusing on the synthesis of the H_∞ controller in this section.

First, let the discrete-time LTI plant $\mathcal{P}$ be a Markov jump linear system of the form

$$\begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{z}_k \\ \boldsymbol{y}_k \end{bmatrix} = \begin{bmatrix} A_{1,\boldsymbol{\theta}_k} & B_{1,\boldsymbol{\theta}_k} & B_{2,\boldsymbol{\theta}_k} \\ C_{1,\boldsymbol{\theta}_k} & D_{11,\boldsymbol{\theta}_k} & D_{12,\boldsymbol{\theta}_k} \\ C_{2,\boldsymbol{\theta}_k} & D_{21,\boldsymbol{\theta}_k} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{d}_k \\ \boldsymbol{u}_k \end{bmatrix} \tag{4.9}$$

Here,

$\boldsymbol{x}_k \in \mathbb{R}^{n_x}$ is the state,

$\boldsymbol{d}_k \in \mathbb{R}^{n_d}$ is the disturbance,

$\boldsymbol{u}_k \in \mathbb{R}^{n_u}$ is the control input,

$\boldsymbol{z}_k \in \mathbb{R}^{n_z}$ is the error, and

$\boldsymbol{y}_k \in \mathbb{R}^{n_y}$ is the measurement.

The state matrices are functions of a finite-state discrete-time Markov chain switching at time k to a mode $\boldsymbol{\theta}_k \in \mathcal{M} = \{1, \ldots, M\}$ with transition probabilities p_{ij} between modes i and j.[2] In our notation, each mode $\boldsymbol{\theta}_k$ is associated with the set of matrices with $\boldsymbol{\theta}_k$ in the second subscript (e.g., $A_{1,\boldsymbol{\theta}_k}$). The switching between modes is governed by whether a

[2]The term *mode* is used interchangeably with the term *state* to describe $\boldsymbol{\theta}_k$ in the Markov chain. This is to avoid confusion with the term *state* used to describe $\boldsymbol{x}_k$.

packet is delivered through the network ($\boldsymbol{\theta}_k = 2$) or a packet is dropped ($\boldsymbol{\theta}_k = 1$).[3] Hence, there are two modes in our single loop MJLS, meaning $\mathcal{M} = \{1, 2\}$.

It was shown in [47] that for MJLSs, mean-square stability, asymptotic mean-square stability, and exponential mean-square stability are all equivalent, so the term "exponentially mean-square stable" is used to mean all three. Moreover, exponentially mean-square stability implies almost-sure stability in MJLSs, but not the other way around.

Seiler and Sengupta choose to synthesize an H_∞ controller because the synthesized controller not only guarantees exponential mean-square stability but also bounds the gain between the disturbance $\boldsymbol{d}_k$ and the closed-loop system error $\boldsymbol{z}_k$. This H_∞-gain of the system is a measure of system performance. A precise definition of the H_∞-gain / H_∞-norm of a MJLS follows from these definitions:

Definition 4.1.1 (Square Summable Sequences)**.** Let $\boldsymbol{x} \triangleq \{\boldsymbol{x}_k\}_{k=0}^{\infty}$ and $\boldsymbol{\Theta}_k \triangleq \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k\}$. Then ℓ_2^n is defined as the *space of square summable (stochastic) sequences*

$$\ell_2^n \triangleq \Big\{\{\boldsymbol{x}_k\}_{k=0}^{\infty} : \forall k, \boldsymbol{x}_k \in \mathbb{R}^n \text{ is a random variable} \\ \text{depending on } \boldsymbol{\Theta}_k \text{ and } \|\boldsymbol{x}\|_2 < \infty\Big\} \tag{4.10}$$

where the ℓ_2-norm $\|\cdot\|_2$ is defined by

$$\|\boldsymbol{x}\|_2^2 \triangleq \boldsymbol{x}_0^{\mathsf{T}}\boldsymbol{x}_0 + \sum_{k=1}^{\infty} \mathop{\mathbb{E}}_{\boldsymbol{\Theta}_k}\left[\boldsymbol{x}_k^{\mathsf{T}}\boldsymbol{x}_k\right] \tag{4.11}$$

and $\mathop{\mathbb{E}}_{\boldsymbol{\Theta}_k}[\cdot]$ denotes taking the expectation over the set of random variables $\boldsymbol{\Theta}_k$. □

Definition 4.1.2 (H_∞-norm)**.** Let $\mathcal{P}$ be an exponentially mean-square stable system and let the initial state $\boldsymbol{x}_0 = 0$. Then the *H_∞-norm*, denoted as $\|\mathcal{P}\|_\infty$, is defined as

$$\|\mathcal{P}\|_\infty \triangleq \sup_{\boldsymbol{\theta}_0 \in \mathcal{M}} \sup_{0 \neq \boldsymbol{d} \in \ell_2^{n_d}} \frac{\|\boldsymbol{z}\|_2}{\|\boldsymbol{d}\|_2} \tag{4.12}$$

□

Seiler and Sengupta synthesize an H_∞ controller $\mathcal{K}$ of the form

$$\begin{bmatrix} \boldsymbol{x}_{C,k+1} \\ \boldsymbol{u}_k \end{bmatrix} = \begin{bmatrix} A_{C,\boldsymbol{\theta}_k} & B_{C,\boldsymbol{\theta}_k} \\ C_{C,\boldsymbol{\theta}_k} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{C,k} \\ \boldsymbol{y}_k \end{bmatrix}. \tag{4.13}$$

[3] Note the slight variation in the usage of $\boldsymbol{\theta}_k$ from Section 4.1.1.

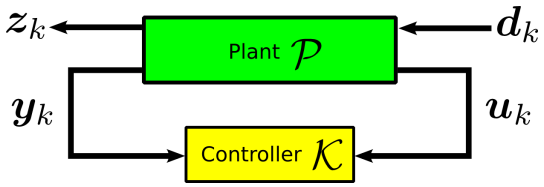

Figure 4.4. Closing the loop for the H_∞ control problem. Here, $\mathcal{P}$ is the plant and $\mathcal{K}$ is the controller and the closed loop system is $F_L(\mathcal{P},\mathcal{K})$. (Figure after [92].)

Note here that the first subscript C is used to distinguish the controller matrices from the plant matrices. The closed loop system is denoted $F_L(\mathcal{P},\mathcal{K})$ and depicted in Figure 4.4.

The main result is stated in the following theorem, where $p_{ij} = p_j$ for all $i,j \in \mathbb{N}_1$ (Bernoulli packet loss model).

Theorem 4.1.2 (From [92])**.** *There exists matrices*

$$0 < Z = Z^{\mathsf{T}} \in \mathbb{R}^{(n_x+n_c)\times(n_x+n_c)} \qquad \textit{and} \qquad \left.\begin{array}{l} A_{C,i} \in \mathbb{R}^{n_c\times n_c} \\ B_{C,i} \in \mathbb{R}^{n_c\times n_y} \\ C_{C,i} \in \mathbb{R}^{n_u\times n_c} \end{array}\right\} \qquad i \in \{1,2\}$$

such that the closed loop system is exponentially mean-square stable and has H_∞-gain less than a user-specified threshold γ if and only if there exists matrices

$$\left.\begin{array}{l} Y = Y^{\mathsf{T}} \in \mathbb{R}^{n_x\times n_x} \\ X = X^{\mathsf{T}} \in \mathbb{R}^{n_x\times n_x} \end{array}\right. \qquad \textit{and} \qquad \left.\begin{array}{l} L_i \in \mathbb{R}^{n_x\times n_y} \\ F_i \in \mathbb{R}^{n_u\times n_x} \\ W_i \in \mathbb{R}^{n_x\times n_x} \end{array}\right\} \qquad i \in \{1,2\}$$

such that

$$\begin{bmatrix} R_{11} & R_{21}^{\mathsf{T}} & R_{31}^{\mathsf{T}} \\ R_{21} & R_{22} & 0 \\ R_{31} & 0 & R_{22} \end{bmatrix} > 0 \tag{4.14}$$

where

$$R_{11} \triangleq \begin{bmatrix} Y & I & 0 \\ I & X & 0 \\ 0 & 0 & \gamma^2 I \end{bmatrix} \qquad R_{22} \triangleq \begin{bmatrix} Y & I & 0 \\ I & X & 0 \\ 0 & 0 & I \end{bmatrix}$$

$$R_{21} \triangleq \sqrt{p_1}\begin{bmatrix} YA_1+L_1C_{2,1} & W_1 & YB_{1,1}+L_1D_{21,1} \\ A_1 & A_1X+B_{2,1}F_1 & B_{1,1} \\ C_{1,1} & C_{1,1}X+D_{12,1}F_1 & D_{11,1} \end{bmatrix}$$

$$R_{31} \triangleq \sqrt{p_2}\begin{bmatrix} YA_2+L_2C_{2,2} & W_2 & YB_{1,2}+L_2D_{21,2} \\ A_2 & A_2X+B_{2,2}F_2 & B_{1,2} \\ C_{1,2} & C_{1,2}X+D_{12,2}F_2 & D_{11,2} \end{bmatrix}.$$

□

Note that (4.14) is a Linear Matrix Inequality (LMI) involving the scalar γ, chosen by the user, and the matrices $Y, X, L_i, F_i,$ and W_i. Checking the condition in Theorem 4.1.2 is equivalent to solving the semidefinite programming problem

$$\begin{aligned} &\min \gamma \\ &\text{subject to (4.14)} \quad . \end{aligned} \tag{4.15}$$

This problem can be solved efficiently by interior-point methods [11] using freely available software [103]. Using the solution to the semidefinite programming problem (4.15), we can construct the controller from the equations

$$\begin{aligned} B_{C,i} &\triangleq Y^{-1}L_i \\ C_{C,i} &\triangleq F_i(Y^{-1} - X)^{-1} \\ A_{C,i} &\triangleq -Y^{-1}[YA_iX + YB_{2,i}F_i + L_iC_{2,i}X - W_i](Y^{-1} - X)^{-1} \quad . \end{aligned} \tag{4.16}$$

4.2 Modeling Unicast Path Diversity

Dust Networks, Inc. designed a networking protocol called the Time Synchronized Mesh Protocol (TSMP) [80] for reliable networking in sensor networks. The protocol exploits frequency, time, and space diversity to achieve what they claim is over 99.9% typical network reliability [29]. TSMP includes a class of protocols which I call *Unicast Path Diversity* (UPD). We propose a Unicast Path Diversity Markov Chain (UPDMC) model to analyze the performance of UPD.[4]

4.2.1 Modeling Characteristics

UPD is a class of frequency-hopping, TDMA, multi-path networking protocols. UPD uses many-to-one routing, i.e., there is one sink in the network. Each node in the network has multiple downstream nodes and the routing topology has no cycles (i.e., is a DAG). The links selected for routing are bidirectional, and hence every link transmission can be acknowledged. If a packet transmission is not acknowledged, it is queued in the node for

[4]The UPDMC model was called the Mesh TDMA Markov Chain (MTMC) model in [15; 16].

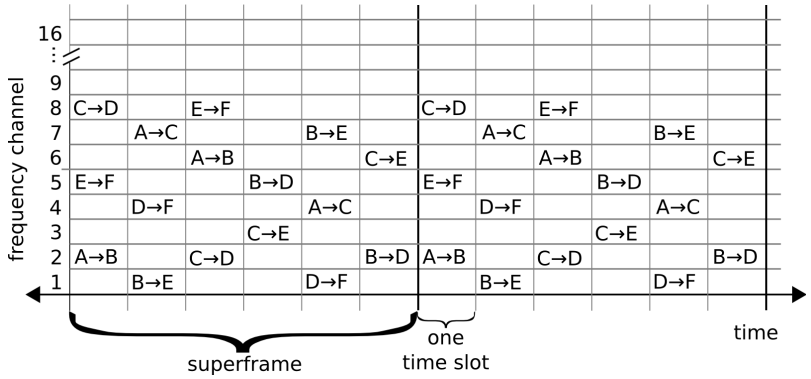

Figure 4.5. Example of a UPD schedule with superframes and time slots. Here, only 8 of the 16 available frequency channels are used.

retransmission. As for scheduling, time is divided into time slots, and grouped into *superframes* (See Figure 4.5). At each time slot, pairs of nodes are scheduled for transmitting a packet on different frequencies if the links are within range to interfere with each other. The superframe containing the schedule of transmissions is repeated over time. Our model uses frequency hopping to justify the assumption that links are independent over retransmissions.

To construct the UPDMC model, we assume knowledge of the routing topology, schedule, and all the link probabilities. Furthermore, we model a single packet transmission in the network and do not analyze the effects of queuing.

4.2.2 Unicast Path Diversity Markov Chain Model

Let us represent the routing topology as a graph $G = (\mathcal{V}, \mathcal{E})$, and denote a node in the network as $i \in \mathcal{V} = \{1, \ldots, N\}$, and a link in the network as $l \in \mathcal{E} \subseteq \{(i,j) \ : \ i,j \in \mathcal{V}\}$, where $l = (i,j)$ is a link for transmitting packets from node i to node j. Time t will be measured in units of time slots, and let T denote the number of time slots in a superframe. The link success probability for link $l = (i,j)$ at time slot t is denoted $p_l^{(t)}$, or $p_{ij}^{(t)}$. We set $p_l^{(t)} = 0$ when link l is not scheduled to transmit at time t.

For a packet originating from a source node a routed to a sink node b, we wish to compute $p_{\text{net}}^{(t_d)}$, the probability the packet reaches b at or before time t_d has elapsed. This is done by a time-varying, discrete-time Markov chain.

Definition 4.2.1 (Unicast Path Diversity Markov Chain Model)**.** Let the set of states in the

Markov chain be the nodes in the network, $\mathcal{V}$. The transition probability from state i to state j at time t is simply $p_{ij}^{(t)}$, with $p_{ii}^{(t)} = 1 - \sum_{j \neq i} p_{ij}^{(t)}$. Let $P^{(t)} = [p_{ij}^{(t)}]^{\mathsf{T}} \in [0,1]^{N \times N}$ be the column stochastic transition probability matrix for a time slot and $P^{(\underline{T})} = P^{(T)} P^{(T-1)} \cdots P^{(1)}$ be the transition probability matrix for a repeating superframe.[5] Assume

$$P^{(T+h)} = P^{(cT+h)}, \qquad \forall c \in \mathbb{N}, h \in \mathbb{N}_1 \tag{4.17}$$

meaning that the link probabilities in a time slot do not vary over superframes.

A packet originating at node a is represented by $\boldsymbol{p}^{(0)} = \boldsymbol{e}^{[a]}$, where $\boldsymbol{e}^{[a]}$ is an elementary vector with the a-th element equal to 1 and all other elements equal to 0. Then,

$$\begin{aligned} \boldsymbol{p}^{(t_d)} = P^{(t_d)} \cdots P^{(2T+1)} \underbrace{P^{(2T)} P^{(2T-1)} \cdots P^{(T+1)}}_{P^{(\underline{T})}} \cdot \\ \underbrace{P^{(T)} P^{(T-1)} \cdots P^{(1)}}_{P^{(\underline{T})}} \boldsymbol{p}^{(0)} \end{aligned} \tag{4.18}$$

represents the probability distribution of the packet over the nodes at time t_d. □

The sink node b is an absorbing state in the Markov chain, meaning there are no transitions out of that state. This means $p_{\text{net}}^{(t_d)} = \boldsymbol{p}_b^{(t_d)}$, the b-th element of the vector $\boldsymbol{p}^{(t_d)}$. One of the characteristics of a good schedule is that $p_{\text{net}}^{(t_d)} \xrightarrow{t_d \to \infty} 1$, meaning the packet will eventually reach the sink. This condition is satisfied when the UPDMC model has only one recurrent class consisting of the sink (See [7] for an explanation and background on Markov chains).

[5] $[0,1]$ denotes the closed interval between 0 and 1. $[\,\cdot\,]^{\mathsf{T}}$ denotes the transpose of a matrix while $P^{(T)}$ denotes the transition matrix at time T.

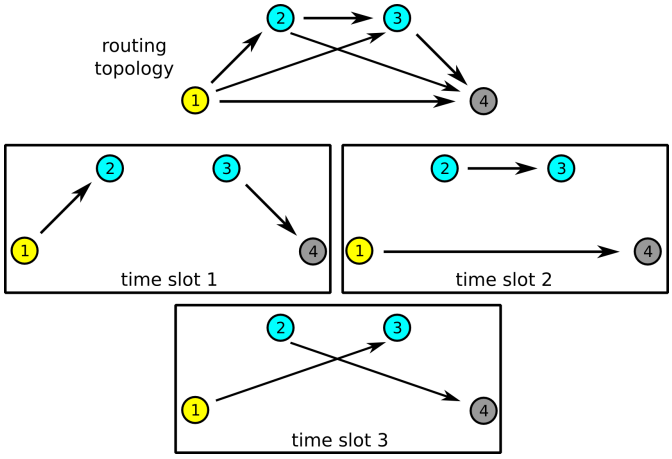

Figure 4.6. Multi-path routing example corresponding to (4.19).

4.2.3 UPDMC Example and Discussion

Example 4.2.1 (UPDMC on a UPD network). An example of a small UPD schedule is given in Figure 4.6, where p_{ij} is the link probability for link (i, j) and $\bar{p}_{ij} = 1 - p_{ij}$. We get the transition probability matrices,

$$P^{(1)} = \begin{bmatrix} \bar{p}_{12} & 0 & 0 & 0 \\ p_{12} & 1 & 0 & 0 \\ 0 & 0 & \bar{p}_{34} & 0 \\ 0 & 0 & p_{34} & 1 \end{bmatrix} \quad P^{(2)} = \begin{bmatrix} \bar{p}_{14} & 0 & 0 & 0 \\ 0 & \bar{p}_{23} & 0 & 0 \\ 0 & p_{23} & 1 & 0 \\ p_{14} & 0 & 0 & 1 \end{bmatrix}$$

$$P^{(3)} = \begin{bmatrix} \bar{p}_{13} & 0 & 0 & 0 \\ 0 & \bar{p}_{24} & 0 & 0 \\ p_{13} & 0 & 1 & 0 \\ 0 & p_{24} & 0 & 1 \end{bmatrix} \tag{4.19}$$

$$\boldsymbol{p}^{(0)} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^{\mathsf{T}} \qquad P^{(\underline{3})} = P^{(3)} P^{(2)} P^{(1)}$$

□

The UPDMC model can be modified to represent unconventional routing topologies and schedules. For instance, the UPDMC model can be extended to represent mesh networks with multiple sink nodes (e.g., two Internet gateways to a sensor network). In this case,

if we let $\mathcal{B}$ be the set of sinks, $p_{\text{net}}^{(t_d)} = \sum_{i \in \mathcal{B}} \boldsymbol{p}_i^{(t_d)}$. Also, the UPDMC model can be used without modification to represent routing topologies with cycles. Typically, a good routing topology will not have cycles but routing cycles may form when the network malfunctions. The UPDMC model can be used to calculate $p_{\text{net}}^{(t_d)}$, where we still have $p_{\text{net}}^{(t_d)} \xrightarrow{t_d \to \infty} 1$ if no recurrent classes besides the sink are added to the Markov chain. Finally, if we wish to model schedules that never retransmit packets, we simply remove the requirement in our UPDMC model that $p_{ii}^{(t)} = 1 - \sum_{j \neq i} p_{ij}^{(t)}$, instead replacing it with $p_{ii}^{(t)} = 0$. To ensure that the transition probability matrix $P^{(t)}$ is a column stochastic matrix, we add a dummy state $N+1$ to represent a packet being lost after transmission. Now, $P^{(t)} = [p_{ij}^{(t)}]^{\mathsf{T}} \in [0,1]^{(N+1)\times(N+1)}$, where $p_{i(N+1)}^{(t)} = 1 - \sum_{j \neq i} p_{ij}^{(t)}, \quad p_{(N+1)i}^{(t)} = 0$ for all $i \neq N+1, \quad$ and $p_{(N+1)(N+1)}^{(t)} = 1$.

4.2.4 UPDMC Analysis

Network-wide Rate of Convergence for $p_{\text{net}}^{(t_d)}$

Besides calculating $p_{\text{net}}^{(t_d)}$ for one node transmitting to the sink, we would like to calculate the rate of convergence of $p_{\text{net}}^{(t_d)} \to 1$ for the *entire* network from $P^{(\underline{T})}$. This may be a better metric for comparing routing topologies and schedules when the network has multiple sessions sharing the same topology and schedule.

Theorem 4.2.1 (UPDMC $p_{\text{net}}^{(t_d)}$ converges exponentially to 1)**.** *Let $P^{(\underline{T})} \in [0,1]^{N\times N}$ be a diagonalizable, column stochastic matrix with $\lim_{k\to\infty}(P^{(\underline{T})})^k \boldsymbol{p} = \boldsymbol{e}^{[b]}$ for all probability vectors $\boldsymbol{p}$. Here, $\boldsymbol{e}^{[b]}$ is an elementary vector with the b-th element equal to 1 and all other elements equal to 0, meaning that the routing topology has a unique sink node b which is the unique recurrent state in the Markov chain. Then,*

$$p_{\text{net}}^{(t_d)} \geq 1 - C(\rho_*)^k, \quad k = \left\lfloor \frac{t_d}{T} \right\rfloor \tag{4.20}$$

for some constant C dependent on the initial distribution $\boldsymbol{p}^{(0)}$ and $\rho_* = \max\{|\lambda| : \lambda$ is an eigenvalue of $P^{(\underline{T})}$ and $|\lambda| < 1\}$. □

Therefore, $p_{\text{net}}^{(t_d)}$ converges to 1 exponentially with a rate ρ_*. The proof of this can be found in Appendix C.1. The rate ρ_* gives a sense of how the end-to-end connection

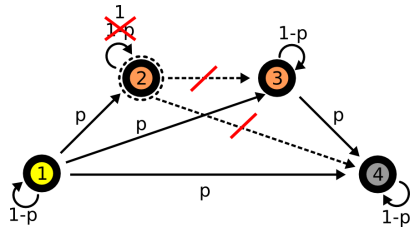

Figure 4.7. Illustration of how to create absorbing states in the Markov chain to calculate the probability that a packet sent from node 1 to node 4 passes through node 2 by time t, using the routing topology of Figure 4.6.

probability to the sink varies as a function of delay in the *worst case* over all nodes In the network.

Traffic Distribution

To identify "hot spots" in the network, we compute the probability that the packet visits a node i *at or before* time t. This can be done by making i an absorbing state in the UPDMC model and finding $\boldsymbol{p}_i^{(t)}$ on the new model.

In other words, $\forall t \in \mathbb{N}_1, \forall j \in \mathcal{V}$, let

$$\begin{aligned}
\tilde{P}_{ji}^{(t)} &= 0 \\
\tilde{P}_{ii}^{(t)} &= 1 \\
\tilde{P}_{uv}^{(t)} &= P_{uv}^{(t)} \quad \forall u, v \in \mathcal{V}, v \neq i
\end{aligned}$$

(See Figure 4.7). The resulting model has two absorbing states, b and i. $\alpha_i^{(t)} = \tilde{\boldsymbol{p}}_i^{(t)} = \tilde{P}^{(t)} \cdots \tilde{P}^{(1)} \boldsymbol{p}^{(0)}$ is the probability that the packet visits node i in the original model at or before time t, while $\alpha_b^{(t)} = \tilde{\boldsymbol{p}}_b^{(t)}$ is the probability that the packet arrives at the sink at or before time t through an alternate path not containing node i.

To find $\alpha_i = \lim_{t\to\infty} \tilde{\boldsymbol{p}}_i^{(t)}$, the probability the packet ever visits node i, we solve a system of equations for the probability that the Markov chain reaches and stays at state i assuming it started from state $j \neq i$.

Theorem 4.2.2 (Absorption probability equations [7])**.** *For a given Markov chain, choose an absorbing state i. Then, the probabilities α_{ji} of reaching state i starting from j are the*

unique solution to the equations

$$\begin{aligned} &\alpha_{ii} = 1 \\ &\alpha_{ji} = 0 \quad \textit{for all absorbing } j \neq i \\ &\alpha_{ji} = \sum_{k=1}^{N} p_{jk}\alpha_{ki} \quad \textit{for all transient } j \quad . \end{aligned} \tag{4.21}$$

□

If a packet is transmitted from a source node a, then $\alpha_i = \alpha_{ai}$.

Sensitivity to Link Estimation Error

Sometimes, we only know that the *actual* probability of a link, $\hat{p}_{ij}^{(t)}$, lies within an interval $(p_{ij}^{(t)} + \epsilon, p_{ij}^{(t)} - \epsilon)$, and estimate it as $p_{ij}^{(t)}$. Unfortunately, we *cannot* bound the range of the actual end-to-end connectivity $\hat{p}_{\text{net}}^{(t_d)}$ by simply recomputing $p_{\text{net}}^{(t_d)}$ using the endpoints $p_{ij}^{(t)} + \epsilon$ and $p_{ij}^{(t)} - \epsilon$. This is because our routing scheme retransmits a packet when a link transmission fails, causing $p_{\text{net}}^{(t_d)}$ to be a polynomial function of the link probability. As a result, $p_{\text{net}}^{(t_d)}$ does not vary monotonically with $p_{ij}^{(t)}$, meaning that $p_{\text{net}}^{(t_d)}$ may not be maximized or minimized by using $p_{ij}^{(t)} \pm \epsilon$ in its calculation.

To see this, let the actual link probability of a link at time slot t be $\hat{p}_{ij}^{(t)} = p_{ij}^{(t)} + \delta$, where δ is unknown to the user but the user knows that $|\delta| < \epsilon$. We can write the actual end-to-end transition probability matrix as

$$\begin{aligned} \hat{P}^{(\underline{T})} &= P^{(T)} \cdots P^{(t+1)}(P^{(t)} + \delta E^{(t)})P^{(t-1)} \cdots P^{(1)} \\ &= P^{(\underline{T})} + \delta \underbrace{P^{(T)} \cdots P^{(t+1)} E^{(t)} P^{(t-1)} \cdots P^{(1)}}_{F} \end{aligned} \tag{4.22}$$

where $E^{(t)}$ is a matrix with -1 at $E_{ii}^{(t)}$, 1 at $E_{ji}^{(t)}$, and 0 elsewhere. Here, for simplicity, we assume that a link is used only once in a superframe. Define $\hat{\boldsymbol{p}}^{(t)}$ as the actual probability distribution at time t. Then,

$$\begin{aligned} \hat{\boldsymbol{p}}^{(T)} - \boldsymbol{p}^{(T)} &= (\hat{P}^{(\underline{T})} - P^{(\underline{T})})\boldsymbol{p}^{(0)} \\ &= \delta F \boldsymbol{p}^{(0)} \end{aligned} \tag{4.23}$$

and

$$\begin{aligned}\hat{\boldsymbol{p}}^{(2T)} - \boldsymbol{p}^{(2T)} &= (\hat{P}^{(\underline{2T})} - P^{(\underline{2T})})\boldsymbol{p}^{(0)} \\ &= (\delta(P^{(\underline{T})}F + FP^{(\underline{T})}) + \delta^2 F^2)\boldsymbol{p}^{(0)} \end{aligned} \tag{4.24}$$

Note that Equation 4.24 is a quadratic function of δ. Thus, because of retransmissions on links (manifested by repeating superframes), it is not clear that the actual $p_{\text{net}}^{(t_d)}$ would vary monotonically with $p_{ij}^{(t)}$.

The alternative is to try bounding the distance of the eigenvalues $\hat{\lambda}$ of the *actual* transition probability matrix $\hat{P}^{(\underline{T})}$ from the eigenvalues λ of our estimated transition probability matrix $P^{(\underline{T})}$, a standard problem in matrix perturbation analysis. In other words, if $\hat{\lambda}_x$ is an eigenvalue of $\hat{P}^{(\underline{T})} = P^{(\underline{T})} + \delta F$, $\delta \in (-\epsilon, +\epsilon)$ and F is a matrix corresponding to the perturbed link ($F_{ii} = -1$, $F_{ji} = 1$, and all other entries are 0), then there is some eigenvalue λ_y of $P^{(\underline{T})}$ such that $|\hat{\lambda}_x - \lambda_y| < C$, where $C \in \mathbb{R}_+$ is a constant that depends only on F and ϵ. There are several standard techniques to do this, some that require $P^{(\underline{T})}$ to be diagonalizable or $P^{(\underline{T})}$ to be normal ($A^*A = AA^*$), which may not always hold. These techniques are applicable on a case by case basis. For more details, see [44].

Energy Consumption

The UPDMC model can also be used to calculate the weighted, expected number of transmissions and receptions up to time t made by a node in the network to transmit a single packet. If we weight the transmit (receive) count by the energy consumption per transmission (reception), U^{Tx} (U^{Rx}), we get the expected radio energy consumption by node i up to time t to relay packets, $\theta_i^{(t)}$. This, together with the baseline energy consumption of the node (energy for sensing and routine operations), will be the overall energy consumption of the node. In fact, in current sensor network platforms the energy spent by the radio is several orders of magnitude greater than the energy spent by the microprocessor (e.g., comparing the CC2420 radio chip and the MSP430 microprocessor when they have the same supply voltage, the ratio of energy consumption is $18.8\,\text{mA}/330\,\mu\text{A} \approx 57$ if the

microprocessor is active and $18.8\,\mathrm{mA}/1.1\,\mu\mathrm{A} \approx 17000$ if the microprocessor is on standby [19; 105]).

To compute $\boldsymbol{\theta}^{(t)} \in \mathbb{R}_+^n$, the vector of $\theta_i^{(t)}$, we construct a vector $\boldsymbol{v}^{(t)} \in \mathbb{R}_+^{2n}$ and a matrix $U^{(t)} \in \mathbb{R}_+^{2n \times 2n}$.

$$\boldsymbol{v}^{(t)} = \begin{bmatrix} \boldsymbol{\theta}^{(t)} \\ \boldsymbol{p}^{(t)} \end{bmatrix} \qquad U^{(t)} = \begin{bmatrix} I & \mathring{I}^{(t)} \\ 0 & P^{(t)} \end{bmatrix} \tag{4.25}$$

where $\mathring{I}^{(t)}$ is a matrix with

$$\begin{aligned} \mathring{I}_{ii}^{(t)} &= \begin{cases} U^{\mathrm{Tx}} & : \quad \exists j \neq i \text{ s.t. } P_{ji}^{(t)} \neq 0 \qquad (i \text{ transmitted at time } t) \\ 0 & : \quad \text{otherwise} \end{cases} \\ \mathring{I}_{ij}^{(t)} &= \begin{cases} U^{\mathrm{Rx}} & : \quad \exists j \neq i \text{ s.t. } P_{ij}^{(t)} \neq 0 \qquad (i \text{ received at time } t) \\ 0 & : \quad \text{otherwise} \end{cases} . \end{aligned} \tag{4.26}$$

$\boldsymbol{\theta}^{(t)}$ can be obtained from $\boldsymbol{v}^{(t)}$, which in turn is computed from

$$\begin{aligned} \boldsymbol{v}^{(t)} &= U^{(t)} \cdots U^{(2)} U^{(1)} \boldsymbol{v}^{(0)} \\ \boldsymbol{v}^{(0)} &= \begin{bmatrix} 0 \\ \boldsymbol{p}^{(t)} \end{bmatrix} . \end{aligned} \tag{4.27}$$

4.3 Modeling Directed Staged Flooding

We define a class of TDMA, constrained flooding protocols called *Directed Staged Flooding* (DSF) for one-to-many and one-to-one routing, focusing on the latter. Unlike UPD, DSF provides increased end-to-end connectivity with less latency by multicasting packets instead of using acknowledgments and retransmissions. We use a Directed Staged Flooding Markov Chain (DSFMC) model to analyze the performance of DSF.

4.3.1 Modeling Characteristics

Like UPD, DSF assumes the routing topology is a DAG and the nodes follow a TDMA schedule. The schedule specifies a transmitting set of nodes for each time slot, where each

node in the transmitting set multicasts to a subset of its downstream neighbors. Multicasting means there will be multiple copies of a packet in the network, so we make a distinction between a *session packet*, which is a packet from the point of view of a session, and *network packets*, which are copies of a session packet sent through the underlying network. In DSF, each node only keeps one copy of a session packet even if it receives the same packet multiple times from upstream nodes. Thus, there can only be multiple network packets when multiple nodes have a copy of the session packet. We assume each node has only one radio, so although a node can multicast to multiple downstream nodes in one time slot, it can only receive from one upstream node in one time slot. A pair of nodes is schedule to transmit on different frequencies in a time slot when the transmission of one node interferes with the reception of the other node's receiving nodes.

DSF does not use acknowledgments to signal a node to retransmit a network packet on a failed link. A network packet is erased from a node after it has been transmitted, regardless of whether the transmission was successful. Thus, with careful scheduling, consecutive session packets will not queue in the network if there is only a single source transmitting to a single sink (but queuing can still happen when there are multiple sessions).

Our DSFMC model groups the nodes along a session's end-to-end transmission paths into groups of nodes called *stages*. Instead of modeling at the level of network packets and nodes, the DSFMC model models a session packet being relayed between stages. Figure 4.8 illustrates this on a *wide path* topology between a source and sink where the nodes lie on a regular grid and each stage, except the first and last, consists of a column of 3 nodes. Each stage also has a *transmitting subset*, which is a subset of nodes in the stage that transmit to the next stage. The transmitting subset of a stage is the set containing all the nodes in a stage unless nodes are shared between stages, which will be discussed later below.

For simplicity, our DSFMC model assumes that a node receives from all its upstream neighbors before it is scheduled to transmit.[6] Given a DSF schedule, one can group the nodes into stages and their transmitting subsets such that all the nodes in the transmitting

[6]This assumption was not in the original DSF model proposed in [15; 16]. Therefore, the stage grouping of the example described by (4.31) and Figure 4.9 has changed.

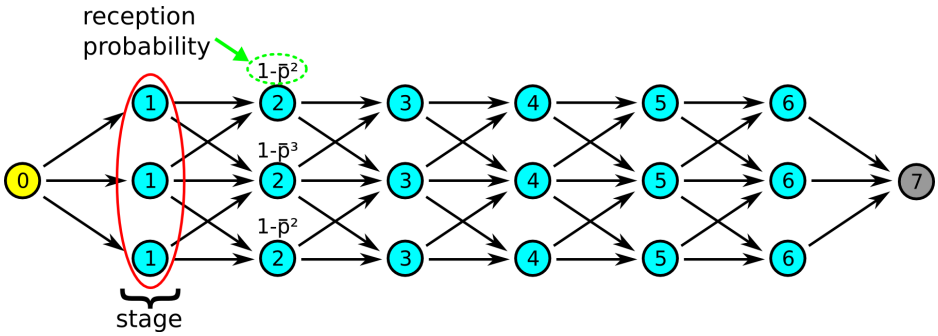

Figure 4.8. Directed Staged Flooding example on a wide path topology with a path width of 3. This example is discussed in more detail in Section 4.3.3.

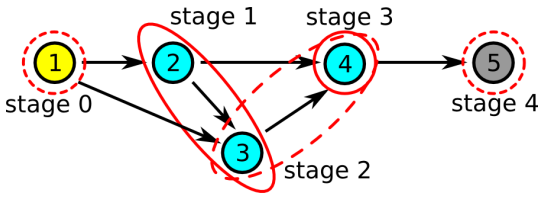

Figure 4.9. Directed Staged Flooding example corresponding to (4.31).

subset of a stage are scheduled to transmit before any nodes in the next stage are scheduled to transmit. This last property is necessary for our DSFMC model. The DSF schedule and the grouping of nodes into stages should be carefully designed to keep the size of the stages from becoming too large. The size of the stages affects the size of the model's state space.

In fact, if we are given the transmitting subsets for each stage, we can determine the nodes in each stage. The nodes in stage k consist of the nodes that have received a network packet from the transmitting subsets of stages $0, \ldots, k-1$ and which have not yet transmitted. Stage 0 consists of the source node. Therefore, nodes are shared between stages k and $k+1$ if they have received a network packet after stage $k-1$ has transmitted and are not in the transmitting subset of stage k. Figure 4.9 gives an example of nodes shared between stages.

Our DSFMC model does not preclude a node i from receiving network packets from other nodes in the same stage, so long as node i is not in the transmitting subset of the stage. The stages in our DSFMC model must be defined such that a node i in stage k can only transmit a network packet if node i received the packet before any other nodes in stage k are scheduled to transmit. If this condition is violated by node i in stage k, we can

replace stage k with multiple stages (which share node i) such that node i transmits in a different stage from the stage when the other nodes transmit to node i.

Our DSFMC model requires the *set* of link transmissions within a pair of stages (k_1, k_1+1) and the set of link transmissions within a pair of stages (k_2, k_2+1) to be independent. Like UPD, DSF uses frequency hopping over time to help justify this assumption. However, the model allows the individual link transmissions within a *single* pair of stages $(k, k+1)$ to be correlated. This mirrors reality because on any single multicast transmission, all the receiving nodes are listening on the same frequency channel.[7]

As with UPD, we construct the DSFMC model assuming we are provided with a routing topology, schedule, the way nodes are grouped into stages, and all the link probabilities. Note that we do not discuss algorithms for grouping nodes into stages in this dissertation, but algorithms for grouping nodes for routing have been proposed in the literature. For example, an algorithm for grouping nodes into stages was proposed by Dubois-Ferriere in [27] for a class of similar routing protocols, *Anypath Routing*, where a packet is routed by multicasting from each node on the path.

4.3.2 Directed Staged Flooding Markov Chain Model

As before, we represent the routing topology as a graph $G = (\mathcal{V}, \mathcal{E})$ and denote a node in the network as $i \in \mathcal{V} = \{1, \ldots, N\}$ and a link in the network as $l \in \mathcal{E} \subset \{(i,j) : i,j \in \mathcal{V}\}$, where $l = (i,j)$ is a link for transmitting packets from node i to node j. Because each link is used only once when transmitting a single packet, the link success probability for link $l = (i,j)$ is treated as being time-invariant and is denoted p_l, or p_{ij}.

Unlike the UPDMC model, in the DSFMC model a state in the Markov chain at a stage represents the *set* of nodes in the stage that successfully received a copy of the packet. The transition probabilities between the states depend on the joint probability of successful link transmissions between stages. We state the DSFMC model for the special case where the links are all independent. For the general model, see B.

[7] Estimating how the links between a pair of stages is correlated, however, can be challenging.

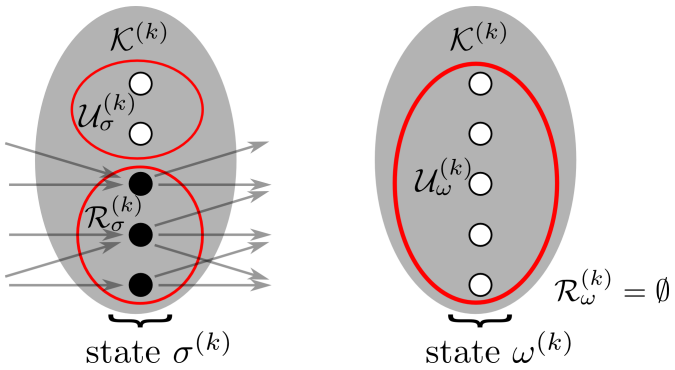

Figure 4.10. Mapping of states to nodes that received a packet in the DSFMC model. On the left is an example of a state $\sigma^{(k)}$ and on the right is the state $\omega^{(k)}$ where no packets have been received.

Definition 4.3.1 (Directed Staged Flooding Markov Chain Model)**.** Let's assume we have a routing topology with $K+1$ stages $0, \ldots, K$. Each stage k has N_k nodes, and the set of 2^{N_k} possible states in stage k is represented by the set of numbers $\mathcal{S}^{(k)} = \{0, \ldots, 2^{N_k} - 1\}$. Let $\mathcal{K}^{(k)}$ be the set of nodes in stage k and for each state $\sigma^{(k)} \in \mathcal{S}^{(k)}$, let $\mathcal{R}_\sigma^{(k)} \subset \mathcal{K}^{(k)}$ be the set of nodes that have received a copy of the packet and $\mathcal{U}_\sigma^{(k)} = \mathcal{K}^{(k)} \backslash \mathcal{R}_\sigma^{(k)}$ be the set of nodes that have not received a copy of the packet (See Figure 4.10). Let $\omega^{(k)}$ denote the state where no nodes received a copy of the packet in stage k.

The conditional probability of the next state $\boldsymbol{X}^{(k+1)}$ being state $\sigma^{(k+1)}$ given that the current state $\boldsymbol{X}^{(k)}$ is $\sigma^{(k)}$ can be expressed as

$$\mathbb{P}(\boldsymbol{X}^{(k+1)} = \sigma^{(k+1)} | \boldsymbol{X}^{(k)} = \omega^{(k)}) = \begin{cases} 1 & : \quad \sigma^{(k+1)} = \omega^{(k+1)} \\ 0 & : \quad \text{otherwise} \end{cases}$$

$$\begin{aligned} &\text{if } \sigma^{(k)} \neq \omega^{(k)} \\ &\mathbb{P}(\boldsymbol{X}^{(k+1)} = \sigma^{(k+1)} | \boldsymbol{X}^{(k)} = \sigma^{(k)}) = \\ &\left(\prod_{\substack{u \in \mathcal{U}_\sigma^{(k+1)} \\ i \in \mathcal{R}_\sigma^{(k)}}} (1 - p_{iu}) \right) \prod_{r \in \mathcal{R}_\sigma^{(k+1)}} \left(1 - \prod_{i \in \mathcal{R}_\sigma^{(k)}} (1 - p_{ir}) \right) \end{aligned} \tag{4.28}$$

The transition probability matrices between stage k and $k+1$ are $P^{(k+1)} \in [0,1]^{N_{k+1} \times N_k}$, where the entry $P_{\sigma^{(k+1)}\sigma^{(k)}}$ of the matrix is $\mathbb{P}(\boldsymbol{X}^{(k+1)} = \sigma^{(k+1)} | \boldsymbol{X}^{(k)} = \sigma^{(k)})$.

The initial state $\boldsymbol{X}^{(0)}$ is the state $\sigma^{(0)}$ corresponding to $\mathcal{R}_\sigma^{(0)} = \{a\}$, where a is the

source node. Then, the probability distribution $\boldsymbol{p}^{(k)} \in [0,1]^{N_k}$ of the state at stage k is

$$\boldsymbol{p}^{(k)} = \underbrace{P^{(k)} \cdots P^{(2)} P^{(1)}}_{P^{(\underline{k})}} \boldsymbol{p}^{(0)} \tag{4.29}$$

□

If we assume only one node in a stage transmits in a time slot and the transmitting subset of each stage is all the nodes in the stage, we can obtain the probability that a copy of the packet is at a node i at time t directly from our model by translating t to k from the relation $t = \sum_{j=0}^{k-1} N_j$ and looking at $\sum_{\{\sigma^{(k)}: i \in \mathcal{R}_\sigma^{(k)}\}} \mathbb{P}(\boldsymbol{X}^{(k)} = \sigma^{(k)})$. In the case where the last stage contains only the sink node and only the nodes in stage $K-1$ transmit to the sink, if b is the state in stage K where the sink receives a copy of the packet, we have

$$p_{\text{net}}^{(t_d)} = \begin{cases} 0 & : \quad t_d \leq \sum_{i=0}^{K-2} N_i \\ \boldsymbol{p}_b^{(K)} & : \quad t_d \geq \sum_{i=0}^{K-1} N_i \end{cases} \tag{4.30}$$

and $0 \leq p_{\text{net}}^{(t_d)} \leq \boldsymbol{p}_b^{(K)}$ when $\sum_{i=0}^{K-2} N_i < t_d < \sum_{i=0}^{K-1} N_i$.

Finally, note that except in the special case where there exists a path from the source to the sink with all link probabilities equal to 1, $p_{\text{net}}^{(t_d)} < 1$ for all t_d. All copies of a packet can be lost in the network because we do not use acknowledgments and retransmissions to guarantee a copy of the packet has been delivered.

4.3.3 DSFMC Examples and Discussion

Example 4.3.1 (DSFMC on a wide path topology). As an example, let's consider the stages with 3 nodes in Figure 4.8. Assume the links are independent, that each link has the same transmission success probability p, and let $\bar{p} = 1 - p$. Then, the probability that a node in stage $k+1$ receives a copy of the packet, given the state of stage k, is 1 minus the product of incoming link failure probabilities, as shown in Figure 4.11. The transition probability between states can be obtained by applying (4.28). Figure 4.12 illustrates the transitions out of state 7. The full 8×8 transition matrix is provided in Appendix D.

□

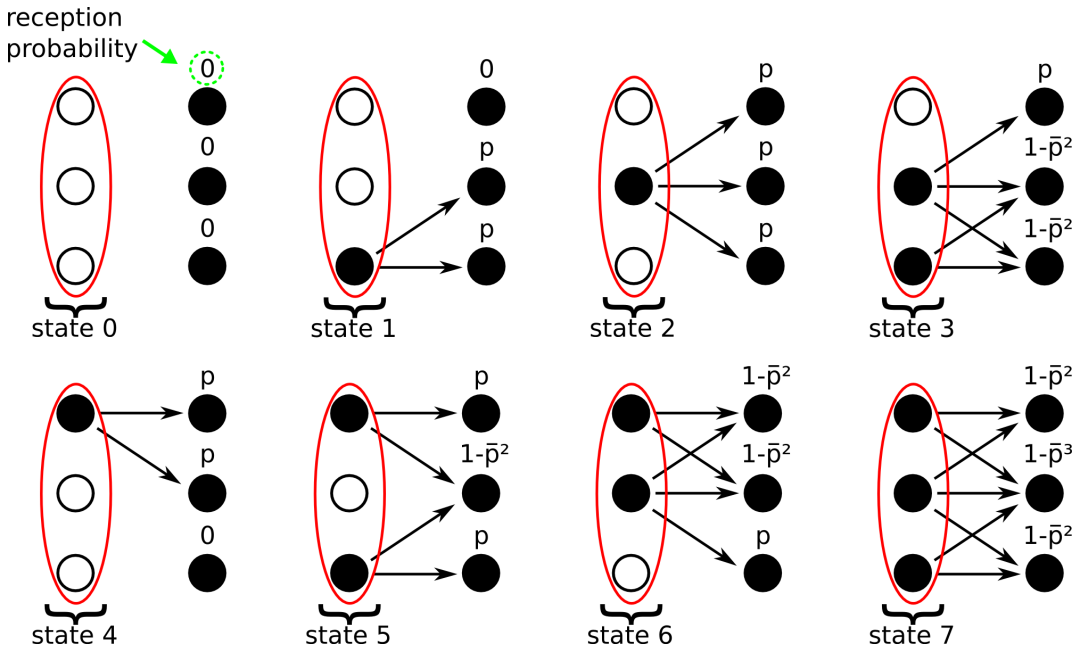

Figure 4.11. Markov chain states for stages with 3 nodes in the routing topology in Figure 4.8.

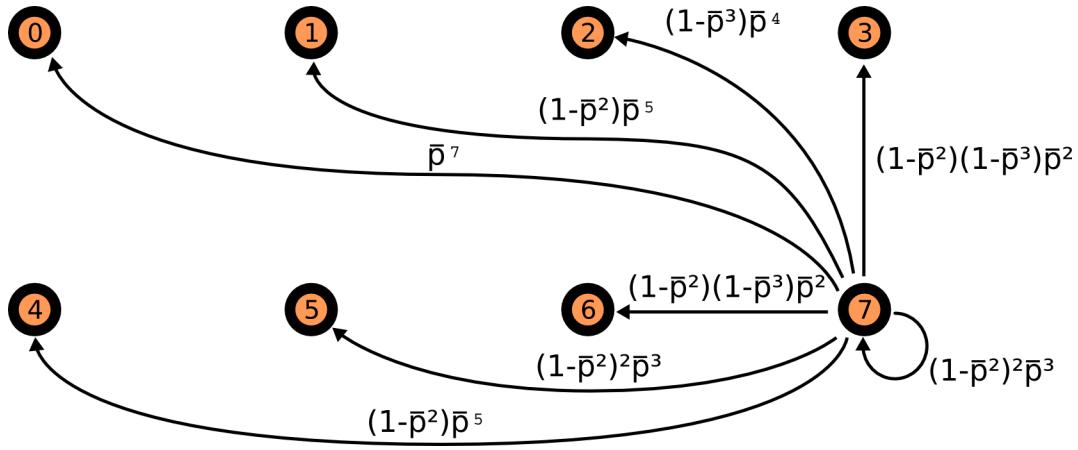

Figure 4.12. Markov chain transition diagram for a stage with 3 nodes in the routing topology in Figure 4.8. Here, only the outgoing transitions and associated transition probabilities from state 7 are shown.

Example 4.3.2 (DSFMC on a topology with shared stages). In the example in Figure 4.9, the dimensions of the state probability distribution vector vary with time, and also some of the nodes are shared between stages. To represent the state at each stage k, we first order the nodes in each stage from smallest to largest node id and re-index them from $0, \ldots, N_k - 1$. Then, for each node with a new index n we define $i_n = 1$ if the node has

a copy of the packet and $i_n = 0$ otherwise. The state is then just $\sigma^{(k)} = \sum_{n=0}^{N_k-1} i_n 2^n$. Assuming the links are independent, the equations that describe the DSFMC model are

$$P^{(1)} = \begin{bmatrix} 1 & \bar{p}_{12}\bar{p}_{13} \\ 0 & p_{12}\bar{p}_{13} \\ 0 & \bar{p}_{12}p_{13} \\ 0 & p_{12}p_{13} \end{bmatrix} \quad P^{(2)} = \begin{bmatrix} 1 & \bar{p}_{23}\bar{p}_{24} & 0 & 0 \\ 0 & p_{23}\bar{p}_{24} & 1 & \bar{p}_{24} \\ 0 & \bar{p}_{23}p_{24} & 0 & 0 \\ 0 & p_{23}p_{24} & 0 & p_{24} \end{bmatrix}$$
$$P^{(3)} = \begin{bmatrix} 1 & \bar{p}_{34} & 0 & 0 \\ 0 & p_{34} & 1 & 1 \end{bmatrix} \quad P^{(4)} = \begin{bmatrix} 1 & \bar{p}_{45} \\ 0 & p_{45} \end{bmatrix} \tag{4.31}$$
$$\boldsymbol{p}^{(0)} = \begin{bmatrix} 0 & 1 \end{bmatrix}^{\mathsf{T}} \quad P^{(\underline{4})} = P^{(4)}P^{(3)}P^{(2)}P^{(1)}$$

where p_{ij} is indexed by the original node ids and again $\bar{p}_{ij} = 1 - p_{ij}$. As mentioned in Section 4.3.1, we assume that if a node i in stage k has a copy of the packet and node i is also in stage $k+1$, then node i will have a copy of the packet in stage $k+1$ with probability 1. □

4.3.4 DSFMC Analysis

$p_{\text{net}}^{(t_d)}$ for Wide Paths with Repeated Stages

For the purposes of choosing a network topology before deployment, it is useful to get a grasp of how $p_{\text{net}}^{(t_d)}$ scales as we extend the length K of a wide path topology without having to calculate $p_{\text{net}}^{(t_d)}$ for each new network explicitly. We consider the case of a wide path with repeated stages containing a constant number of nodes N_{stage} per stage and the same transition probability matrix $P^{(k)} = P$ between all stages, like the middle stages in the example in Figure 4.8. For simplicity, the discussion below will ignore the first stage containing the source and the last stage containing the sink.

A good characterization of how end-to-end connectivity scales with the number of stages K comes from the eigenvalues of P.

Theorem 4.3.1 (DSFMC $p_{\text{net}}^{(t_d)}$ converges exponentially to 0)**.** *Let P be diagonalizable and*

$\lim_{K\to\infty} P^K \boldsymbol{p}^{(0)} = \boldsymbol{e}^{[\omega]}$, *where* ω *is the state where* no *nodes received a copy of the packet. Then*

$$p_{\text{net}}^{(t_d)} \leq C(\rho_*)^K, \quad t_d = KN_{\text{stage}} \tag{4.32}$$

for some constant C *dependent on the initial distribution* $\boldsymbol{p}^{(0)}$ *and* $\rho_* = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } P \text{ and } |\lambda| < 1\}$. □

The proof of this can be found in Appendix C.2. While this relation is an upper bound, ρ_* is the dominant decay rate for large K because all the eigenvectors of P with eigenvalue magnitudes less than 1 decay exponentially with K. In practice, a good routing topology has ρ_* very close to 1. When choosing a routing topology for wide paths, one can use ρ_* for wide paths with repeated stages of different sizes to quickly compare the gain in reliability at the cost of extra latency.

Traffic Distribution

To calculate the probability that a copy of the packet visits a node i at or before time t, $\alpha_i^{(t)}$, we first remove all the outgoing edges of i, and add a "self transmission" link of probability 1 from node i to itself over all time slots. Then, we compute $\alpha_i^{(t)} = \sum_{\{\sigma^{(k)}: i\in\mathcal{R}_\sigma^{(k)}\}} \mathbb{P}(\tilde{\boldsymbol{p}}^{(k)} = \sigma^{(k)})$, where $\tilde{\boldsymbol{p}}^{(k)}$ is the state probability distribution on the modified schedule and topology.

Sensitivity to Link Estimation Error

As with UPD, our DSFMC model is constructed from estimated link probabilities p_l, and the user does not know the actual link probability $\hat{p}_l = p_l + \delta$. In DSF, where each node transmits only once per packet and all links are independent, if the user knows that the actual probability of a link $\hat{p}_l = p_l + \delta$ lies in an interval $(p_l + \epsilon, p_l - \epsilon)$ he can bound the actual end-to-end connectivity $\hat{p}_{\text{net}}^{(t_d)}$ by simply recomputing $\hat{p}_{\text{net}}^{(t_d)}$ using the endpoints of the interval $p_l \pm \epsilon$. This is because unlike UPD, there are no link retransmissions in DSF, which implies that $p_{\text{net}}^{(t_d)}$ is a linear function of the single-link estimation error δ.

To see this, note that in (4.28) the transition probabilities between states in adjacent stages are a linear function of the individual link probabilities (the probability associated with a link appears in the expression once). This means that the transition matrices $P^{(k)}$ are a linear function of each link probability p_l. Also, each link probability p_l appears in only one matrix $P^{(k)}$ because each link is used only once to transmit a packet. This is because each node only transmits once when routing a single packet through the network. As a result, $P^{(\underline{K})}$ is also a linear function of p_l. Finally, $p_{\text{net}}^{(t_d)}$ is a linear function of $P^{(\underline{K})}$ and hence also a linear function of p_l, meaning it is a linear function of δ.

Energy Consumption

As noted in Section 4.2.4, the weighted expected number of transmissions and receptions up to time t by node i for routing a single session packet (and its copies) through the network, $\theta_i^{(t)}$, can be interpreted as the expected radio energy consumption up to time t to relay a session packet. In DSF on topologies with disjoint stages, the expected number of transmissions is simply the probability that a packet reaches node i because there are no acknowledgments and retransmissions. The expected number of receptions at node i is the sum of the probabilities that a packet reaches the nodes that can transmit to node i. The following formulation is still useful for seeing how to formally calculate the energy consumption.

To simplify calculations, we will compute $\theta_i^{(k)}$, the weighted expected number of transmissions at node i and receptions up to the completion of the transmission interval between stage $k-1$ and stage k (The transmission interval is the time interval over which the transmitting subset in stage $k-1$ transmits). Let $\boldsymbol{\theta}^{(k)} \in \mathbb{R}_+^n$ be the vector of $\theta_i^{(k)}$ where k is the current transmitting stage. Let U^{Tx} be the energy consumed per transmission and U^{Rx} be the energy consumed per reception (these are the weighting factors). Define the vector $\boldsymbol{v}^{(k)} \in \mathbb{R}_+^{n+N_k}$ and the matrix $U^{(k)} \in \mathbb{R}_+^{(n+N_k)\times(n+N_{k-1})}$ to be

$$\boldsymbol{v}^{(k)} = \begin{bmatrix} \boldsymbol{\theta}^{(k)} \\ \boldsymbol{p}^{(k)} \end{bmatrix} \qquad U^{(k)} = \begin{bmatrix} I & \mathring{I}^{(k)} \\ 0 & P^{(k)} \end{bmatrix} \tag{4.33}$$

where $\mathring{I}^{(k)} \in \mathbb{R}_{+}^{n \times N_{k-1}}$ has entries defined by

$$\mathring{I}^{(k)}_{i\sigma^{(k-1)}} = \begin{cases} U^{\mathrm{Tx}} & : \quad \exists P_{\sigma^{(k)}\sigma^{(k-1)}} \neq 0 \text{ s.t. } (i \in \mathcal{R}_{\sigma}^{(k-1)} \text{ and} \\ & \quad \exists j \in \mathcal{R}_{\sigma}^{(k)}, j \neq i) \\ & \quad (i \text{ transmitted during} \\ & \quad \text{stage } k-1\text{'s transmission interval.)} \\ m \cdot U^{\mathrm{Rx}} & : \quad \exists P_{\sigma^{(k)}\sigma^{(k-1)}} \neq 0 \text{ s.t. } i \in \mathcal{R}_{\sigma}^{(k)} \\ & \quad m \triangleq \left|\mathcal{R}_{\sigma}^{(k-1)}\right| - \mathbb{I}(i \in \mathcal{R}_{\sigma}^{(k-1)}) \\ & \quad (i \text{ received } m \text{ times during} \\ & \quad \text{stage } k-1\text{'s transmission interval.)} \\ 0 & : \quad \text{otherwise} \end{cases} . \tag{4.34}$$

Note that the expression

$$\exists j \in \mathcal{R}_{\sigma}^{(k)}, j \neq i$$

and the definition

$$m \triangleq \left|\mathcal{R}_{\sigma}^{(k-1)}\right| - \mathbb{I}(i \in \mathcal{R}_{\sigma}^{(k-1)}) \quad ,$$

where $\mathbb{I}$ is the indicator function, are to ensure that we are not counting transmissions from node i to itself.

$\boldsymbol{\theta}^{(k)}$ can be obtained from $\boldsymbol{v}^{(k)}$, which in turn is computed from

$$\boldsymbol{v}^{(k)} = U^{(k)} \cdots U^{(2)} U^{(1)} \boldsymbol{v}^{(0)} \tag{4.35}$$

$$\boldsymbol{v}^{(0)} = \begin{bmatrix} 0 \\ \boldsymbol{p}^{(k)} \end{bmatrix} \quad . \tag{4.36}$$

4.4 UPD and DSF Tradeoffs

This section uses the connectivity metric $p_{\mathrm{net}}^{(t_d)}$ to compare UPD and DSF on the same routing topology to explain some of the tradeoffs between the two networking protocols. The real-time measurement system case study in Section 4.5.1 will use the tools developed

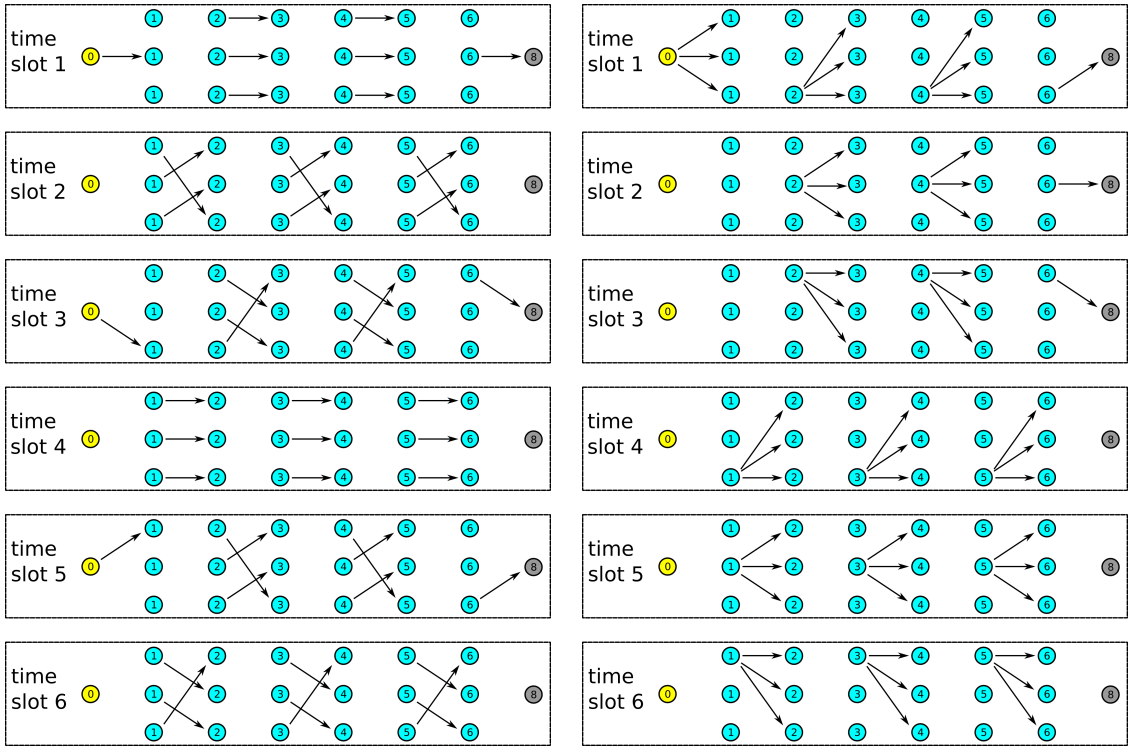

Figure 4.13. (left) UPD and (right) DSF schedules for routing on a width-3 path topology, used in the calculations for Figures 4.14, 4.15, and 4.16.

in this chapter to compare UPD and DSF in a more complex scenario. Recall that UPD uses retransmissions to increase reliability while DSF uses multicast and multiple copies of packets to increase reliability.

We chose the example of routing on a wide path topology, where the width of the path is the number of rows and the length of the path is the number of columns. Here, every node in one column (a stage in DSF) can route to every other node in the next column with equal, independent link probabilities p_l. The schedules for Directed Staged Flooding and Unicast Path Diversity is depicted in Figure 4.13 for a width-3 path topology. Also, in all our comparisons, we assume that the time to send an acknowledgment (ACK) for UPD is negligible and ACKs can be sent back in the same time slot as the original transmission.

Figure 4.14 compares $p_{\text{net}}^{(t_d)}$ of UPD and DSF, calculated using the UPDMC and DSFMC models, under a range of different link probabilities.[8] UPD has the potential to deliver

[8]Note that in this and subsequent plots, we perform the DSFMC calculations at the time granularity of time slots, not stages, unlike the description of (4.30) in Section 4.3.2.

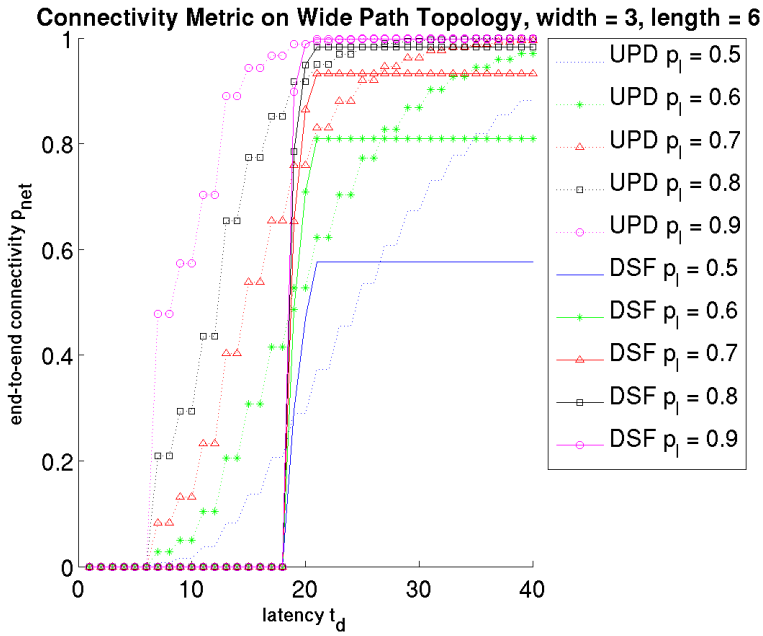

Figure 4.14. End-to-end connectivity as a function of latency for varying link probabilities using the schedules described in Figure 4.13.

packets from the source to the sink in a shorter period of time, but the packet delivery time has a larger variance. The packet delivery time for DSF spans a short, fixed range because the DSF schedule of Figure 4.13 was designed to keep the copies of a packet within a small set of nodes (at most two adjacent columns at any time) such that a node receives copies of the packet from all its upstream neighbors before it transmits. This means that each upstream neighbor of the sink will only transmit the packet once. Since DSF allows us to keep copies of a packet within a small set of nodes, DSF can also avoid queuing in the network (e.g., in our example, if the source generates a new packet every 6 time slots). Also, because $p_{\text{net}}^{(t_d)} \xrightarrow{t_d \to \infty} 1$ for UPD and p_{net} for DSF is a fixed value less than 1 after the last stage transmits (assuming $p_l \neq 1$), UPD can always provide better end-to-end connectivity at high latencies t_d.

Figure 4.15 shows that the final end-to-end connectivity p_{net} for DSF is higher for wider paths at the cost of larger latency. Also, the figure illustrates the limitations of our UPDMC model—the model is unable to capture the benefit of diversity from using multiple paths instead of a single path because it assumes that all links are independent. Hence, retransmission on the same link is just as good as transmitting on a different link. What

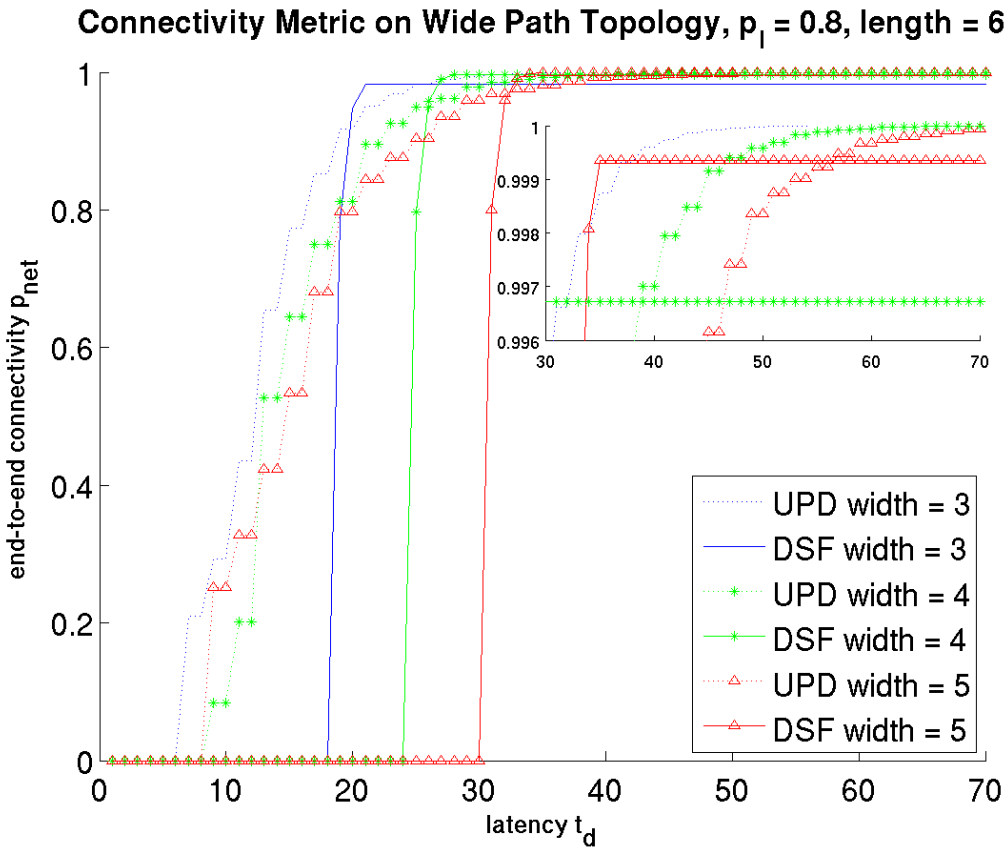

Figure 4.15. Calculated end-to-end connectivity as a function of latency for varying path widths using the schedules described in Figure 4.13, with magnification of the plot for p_{net} near 1.

is modeled is that wider paths require more time slots to schedule transmissions from the nodes in the last column to the sink, explaining why Figure 4.15 shows that UPD on wide paths with a smaller width tend to perform better than UPD on wide paths with a larger width.

Since it is common for routing topology formation algorithms to only use links with probabilities higher than a threshold in their routing topology, it is tempting to use the plots in Figure 4.14 for high link probabilities to conclude that UPD always has better end-to-end connectivity than DSF on wide path topologies even for short latencies t_d . However, wireless channels may exhibit *fading*, causing the link probability on some links to suddenly drop to a very low value. In the scenario where all the outgoing links on a relay node happen to have a low probability because of fading, then a packet can get trapped at the node, as discussed earlier in Section 2.3. In this scenario the link probabilities in the topology will

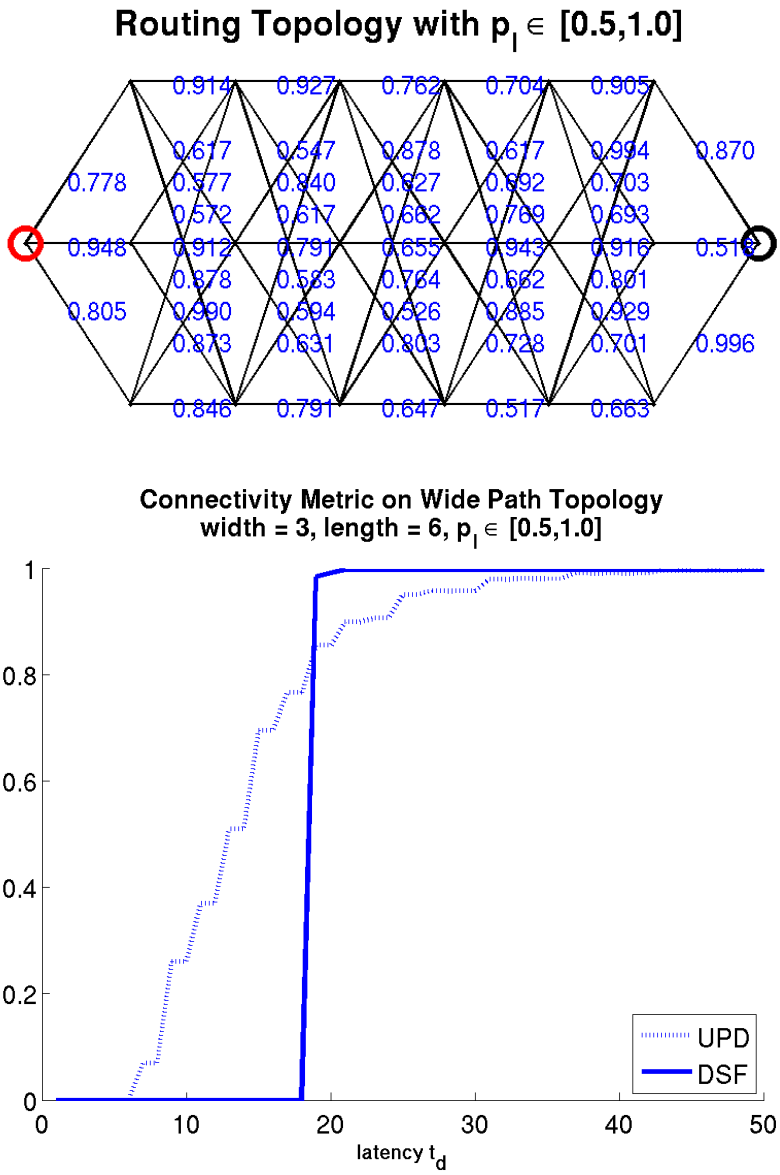

Figure 4.16. Example of a topology with unequal link probabilities, where at short latencies $19 \leq t_d < 48$ it is preferable to use the DSF schedule rather than the UPD schedule in Figure 4.13. (left) Routing topology, links labeled by their link probabilities, which were drawn uniformly at random from the interval $[0.5, 1]$. (right) End-to-end connectivity for UPD and DSF.

not all be equal and DSF may provide a better end-to-end connectivity than UPD at short latencies t_d. Figure 4.15 gives an example where the link probabilities in the width-3 path topology are drawn uniformly at random from the interval $[0.5, 1]$ to simulate a mixture of good and bad links in the network. When $19 \leq t_d < 48$, it is preferable to use DSF.

4.5 Case Studies on Wireless Networked Systems

This section will demonstrate how the models and tools presented in this chapter can be used to help design wireless networked systems. The calculations from the models are used in the design iterations to quickly check the schedules and other network parameter choices and guide the design. It must be stressed, however, that simulations and experiments on real platforms still play an important role in the final design phase to confirm that the assumptions made in our models are reasonable. For instance, our models do not account for queuing in the network, so we tried to reduce queuing effects by staggering the times when packets are sent into the network. We use simulations to confirm that queuing occurs infrequently in the network.

The case studies below are more complex than the previous examples used to illustrate our Markov chain models because they involve multiple sessions and sources that periodically generate data over time. Thus, we will need to define some new terminology for our discussions below. Recall that a *session* $s \in \{1, \ldots, S\}$ is an end-to-end connection between a pair of users. In our scenarios below, each session has one source node a and one sink node b. We say session s has source node a_s, and the vector $\boldsymbol{a} \in \mathbb{N}_1^S$ represents the collection of session to source node mappings. Different sessions may share a routing topology if they have the same sink node. However, all sessions share the same schedule.

We say that a session *injects* a packet into the network when it places the packet into a queue at the source node for transmission through the network. Let the *session send time* for a packet be the time the packet is injected into the network. Each packet has a *Time-to-Live* (*TTL*) counter to keep track of how long the packet has been in the network so it can be removed when it is too old. We say a packet *has a TTL of* t_{TTL} *time slots* when we set the initial value of the TTL counter to t_{TTL}. Starting from the packet's session send time, the counter is decremented on each time slot (regardless of whether the packet is transmitted on a link[9]) until it reaches 0, at which point the packet is removed from the

[9]The packets on the Internet (IPv4) also have TTL counters, but the counters are decremented on each hop and not decremented with time, i.e., counters are not decremented as the packet sits in a queue in the network. In IPv6, the TTL counter is renamed the *hop limit* counter.

network. We will also refer to t_{TTL} as the *packet simulation time*, the number of time slots we simulate a packet in the network.

In DSF, we make a distinction between a *session packet* and *network packets*, because the network may have multiple copies of the same packet injected into the network. The term "session packet" refers to the (unique) packet as seen by the users of the session, while the term "network packet" refers to the (possibly many) copies of the packet being routed through the network. Naturally, we assume that the networking protocols will eliminate duplicate received network packets to present a single session packet to the users. We say a session packet is *cleared* from the network when all of its network packets are removed from the network either because they have been received by the sink or their TTL counter has expired.

The *packet latency* t_d of a session packet is the sum of its *packet wait time* t_w and its *packet relay time* t_r. The packet wait time is the number of time slots the packet waits in the queue at the source node before the source node is scheduled to transmit (regardless of whether the transmission is successful). The *maximum packet wait time* depends on the schedule and depends on when a session can inject packets into the network. The packet relay time (for DSF, this is also called the *minimum packet relay time*) is the number of time slots between when the source is scheduled to transmit and when the first network packet is received at a sink node. In DSF, we define the *maximum packet relay time* as the number of time slots between when the source is scheduled to transmit and when the last network packet is received at a sink node. Figure 4.17 illustrates these concepts and how they are used to calculate the maximum packet latency for DSF.

In our case studies, the schedules are described by repeating a superframe of time slots over time. Let $\mathcal{F}_s$ be a set whose elements are the time slots in the superframe when session s can inject a packet into the network, and call $\mathcal{F}_s$ the *superframe send times* of session s. Let ϑ_s be the superframe send time of the first packet of session s, and $\boldsymbol{\vartheta}$ be the vector of ϑ_s for all the sessions. We can calculate $\mathcal{F}_s$ if session s generates packets periodically. Let $\zeta_s \in \mathbb{N}_1$ be the *packet generation period* of session s, which is the number of time slots between session packets. If the packet generation period is the same for all sessions, we use

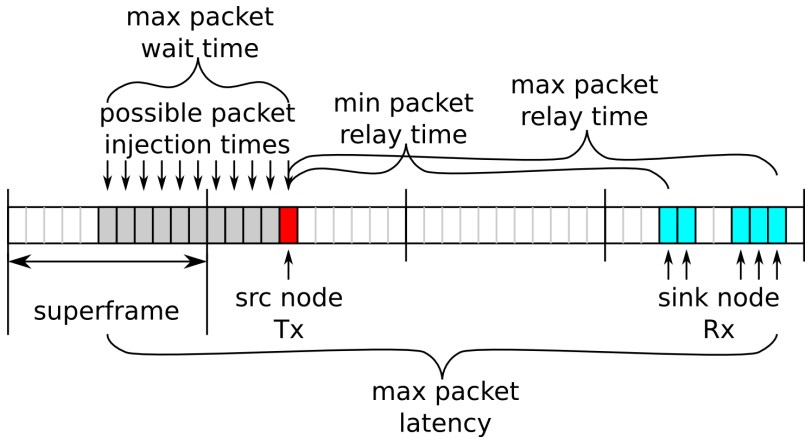

Figure 4.17. Illustration of how to calculate maximum packet latency using DSF session 2 in the real-time measurement system case study (case study 1) as an example. Copies of the packet are received over 5 time slots at the sink node. See Section 4.5.1 for details.

ζ to denote the packet generation period. In our real-time control case study (case study 2), the packet generation period is simply the *system sampling time*, the number of time slots in a sampling period of our networked control system. Then,

$$\mathcal{F}_s = \Big\{ \big((\vartheta_s - 1) + k \cdot \zeta_s \pmod{T} \big) + 1, \quad k = 1, \ldots, T-1 \Big\} \tag{4.37}$$

where T is the length of a superframe.

Many of the preceding terms and concepts will be used in our discussions below to compare and choose schedules.

We will use time constants from WirelessHART [40] and the performance characteristics of the TelosB platform [21] to relate our simulations and calculations to real protocols and platforms. We assume there are 100 time slots per second, like in WirelessHART. Using the voltage and current consumption characteristics of the TelosB during transmission and reception, and assuming (to a rough approximation) that a packet transmission or reception takes the full time slot (10 ms), the energy usage is

$$(3\,\mathrm{V})(21\,\mathrm{mA})(\frac{1}{100}\,\mathrm{sec}) = 630\,\mu\mathrm{J}$$

per packet transmission and

$$(3\,\mathrm{V})(23\,\mathrm{mA})(\frac{1}{100}\,\mathrm{sec}) = 690\,\mu\mathrm{J}$$

per packet reception.

We will use simulations to verify the calculations of $p_{\text{net}}^{(t_d)}$ and our assumption that queuing in the network has a small impact on packet delivery in our scenarios. In our simulations, each node has one queue to hold the packets from all the sessions. When an outgoing link on a node is scheduled to transmit, we look for the packet closest to the top of the queue that can be transmitted on that link (i.e., the routing topology associated with a packet contains that link). In this manner, packets in the queue are not blocked from transmission because the packet at the top of the queue cannot transmit on the link scheduled in the current time slot.

4.5.1 Case Study 1: Real-Time Measurement System

The tools developed in this dissertation can be used to plan and design WSNs for real-time measurement systems before deployment. The designer can quickly check the feasibility of a particular network routing topology and schedule given reasonable assumptions on the link probabilities. Furthermore, the tools can be used for online estimation of network performance if link statistics are collected online to estimate the link probabilities. This section will use our analysis tools to check whether a particular wireless camera network deployment is feasible for building surveillance.

Video Surveillance

Our real-time measurement system case study is building surveillance using wireless camera embedded platforms with significant on-board processing capabilities, such as the CITRIC mote [17]. The CITRIC mote can communicate with other wireless sensor network platforms using IEEE 802.15.4. Furthermore, developers have the flexibility to code or install different networking protocols on the mote, including TDMA mesh-networking protocols that fit the Markov chain models in this dissertation.

In our building surveillance scenario, we will be tracking intruders (targets) that break into a building after-hours using a wireless camera network. As this is not a dissertation on computer vision, we will make several simplifying assumptions on the problem. First,

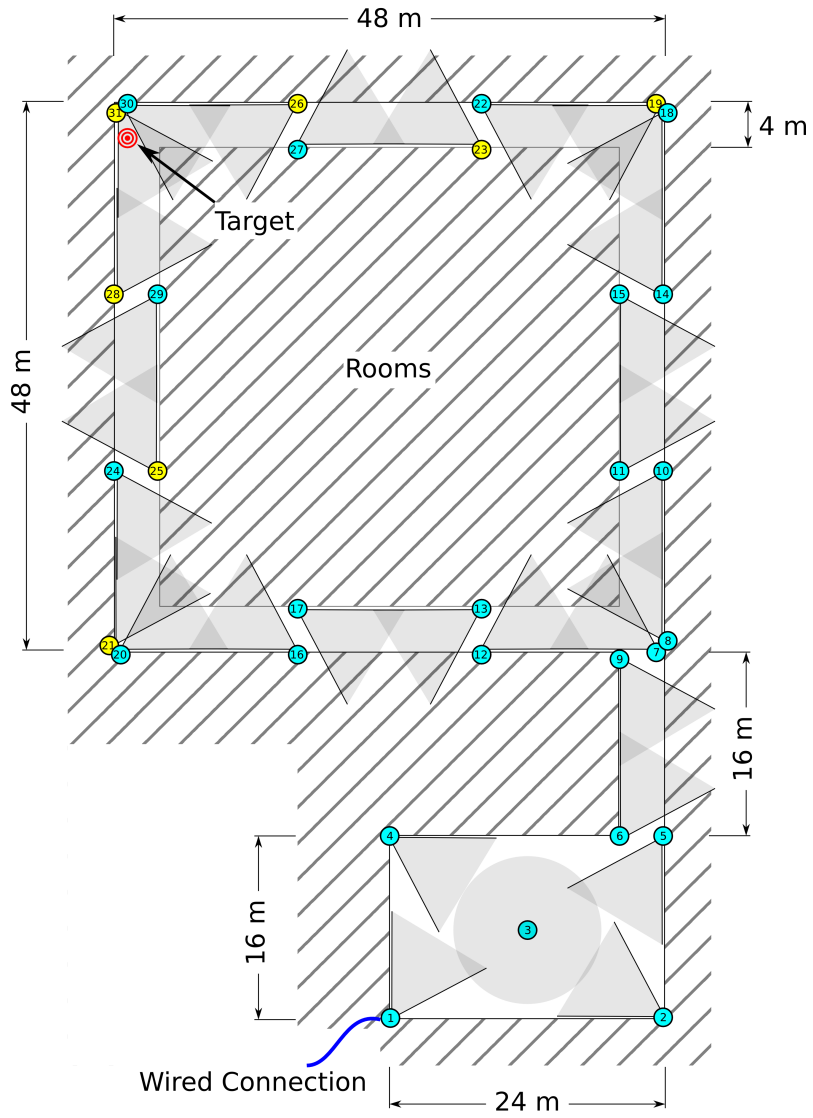

Figure 4.18. Layout of camera nodes to monitor the hallways on a floor of the building. Node 1, the sink node, has a wired connection to a centralized server which serves as the estimator in Figure 1.1 and the interface to the user. The target (in red) is visible by nodes 19, 21, 23, 25, 26, 28, 31 (highlighted in yellow). Each camera has a field-of-view (FOV) of roughly 60°, represented by the gray shaded triangle. The FOV extends out to infinity until hitting an obstruction but is not drawn this way for clarity.

let's assume there is only one target in the building. Second, this case study will focus on monitoring the hallways of a single floor in the building. Third, the cameras are placed such that all locations in the hallway are covered by two or more cameras (See Figure 4.18). This assumption simplifies the handoff of target-tracking information between cameras. Fourth, we assume that the cameras are calibrated beforehand to share a global frame of reference (i.e., we know the Fundamental matrix for each pair of cameras).

Our goal is to check the feasibility of a sample wireless camera network deployment in a building. Under reasonable assumptions of the wireless network conditions, we would like to compute the end-to-end connectivity as a function of latency, the traffic distribution, and the expected energy usage of the nodes in the network.

Wireless Network

Our wireless camera nodes do not stream images over the wireless network. Instead, each camera performs local computations to track the target and pass this tracking information back to a centralized server for global reconstruction of the track (wired node in Figure 4.18). In general, clusters of cameras may communicate locally to extract more information from the scene, but we will not consider this in our case study.[10] In our calculations below, we will assume that each camera sends one packet with tracking information per image (packets have a maximum payload of 118 Bytes in 802.15.4, See Table 1.1).

We consider a "worst case" scenario for our network, where a target is visible by 7 cameras that are located far away from the camera node wired to the centralized server, as depicted in Figure 4.18. Here, we have 7 sessions, one routing from each of these camera nodes to the wired camera node (the sink node). The mapping from session to source node id is $\boldsymbol{a} = [31\ 28\ 26\ 25\ 23\ 21\ 19]$. The cameras capture and process images at the same frame rate, so the packet generation period for all the cameras is the same and is denoted as ζ.

Let's assume each node can communicate with most other nodes within a 35 m range. The exceptions to this rule are communication from nodes 1, 4, and 3 to nodes 12, 13, 16, and 17, because to be within range they must communicate through thick walls and rooms. Most of the edges in the connectivity graph is depicted in the routing topology of Figure 4.19. Edges between nodes in the same column in Figure 4.19 are missing, because they are not used in the routing topology. The routing topology for our network simply

[10] To give an idea of the bandwidth requirements if we communicate image features between cameras, consider SIFT features, which are used in many computer vision algorithms. In [65], Lowe mentions that a typical "500×500 pixel image can give on the order of 2000 features." A typical SIFT descriptor represented by a 128-element vector of 2 Byte numbers is 256 Bytes. A compact representation of SIFT, called PCA-SIFT, with 20-element vectors of 2 Byte numbers is 40 Bytes [50]. If we naïvely sent all the PCA-SIFT features from one camera to another camera for correspondence, we would need to send $\approx 2000 \times 40$ Bytes $= 80$ kB of data per image.

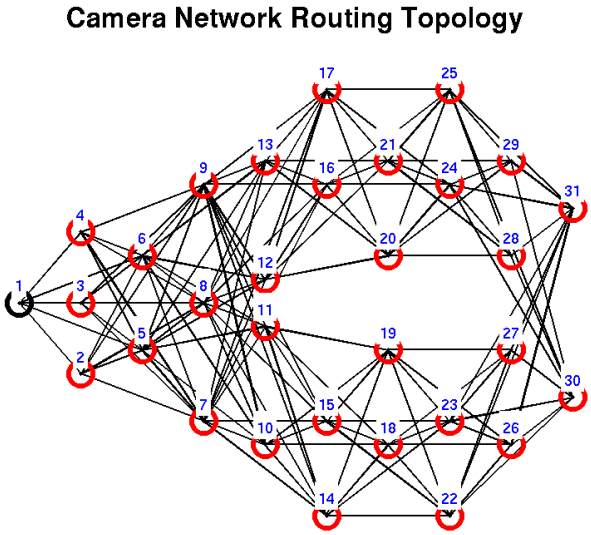

Figure 4.19. Routing topology for the camera network layout in Figure 4.18. For clarity, the position of the nodes have been rearranged and the arrowheads have been removed from the edges. Assume all links are oriented from right to left. Generally, nodes can route to other nodes that are 1 or 2 columns to its left.

orients the edges in Figure 4.19 from right to left. In our scenario, all sessions share this same routing topology. The links between nodes with direct line-of-sight have probability 0.9,[11] and all other links have probability 0.6.

The UPD schedule is a repeating superframe with $T = 14$ time slots, given in Appendix E.1. The links scheduled to transmit during each time slot comprise a maximal matching. Each link is scheduled to transmit at least once during the superframe. We sidestep the frequency assignment problem by allocating a unique frequency to each link transmitting in a time slot (we have 16 frequency channels in the 2.4 GHz band in 802.15.4 and no more than 15 links per time slot).

Likewise, the DSF schedule is a repeating superframe, this time with $T = 11$ time slots and given in Appendix E.2. Each column of nodes in the routing topology is a transmitting subset of a stage, as shown in Figure 4.20. The set of nodes that have a nonzero probability of receiving a network packet before the transmitting subset of stage k is scheduled to transmit is designated as the nodes in stage k. Each node is scheduled to multicast to its downstream

[11] Exceptions: we treat the links (7,3), (8,3), and (9,3) as having line-of-sight.

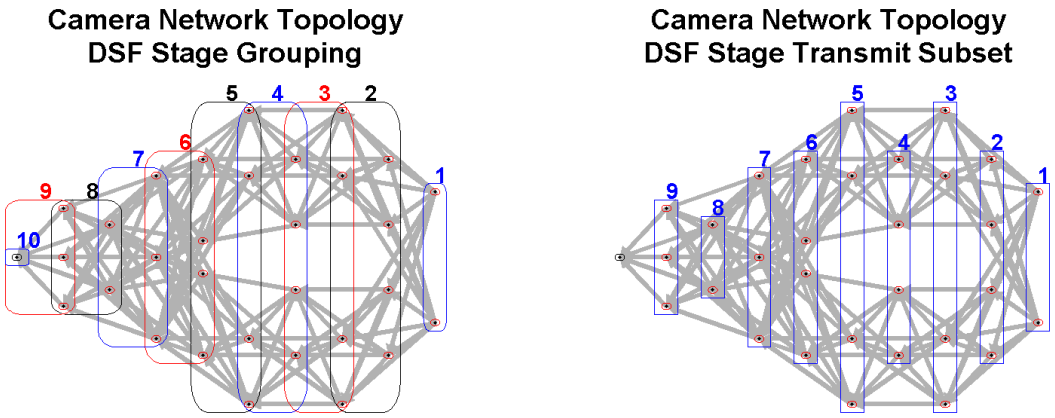

Figure 4.20. Stages in DSF for the camera network routing topology depicted in Figure 4.19. (left) Nodes in each stage are grouped by colored, rounded rectangles. (right) The transmitting subset of each stage are grouped by rectangles.

neighbors exactly once during the superframe. Again, we sidestep the frequency assignment problem by allocating a unique frequency to each link transmitting during a time slot.

Calculations and Simulations

First, we use the UPDMC model to calculate how many time slots it would take for the packets in each session to reach the sink node under UPD. This depends on which time slot the packet is first injected into the network. For our calculations, we set the first superframe send time for session s, ϑ_s, to time slot $2s-1$ so the time that different session packets enter the network are staggered. This is to reduce the amount of queuing in the network, which is not modeled by the UPDMC model. In our UPDMC model (4.18), this means the transition probability matrices for the first session packet are shifted over by $2s-2$ time slots, e.g., $P^{(1)}$ would be replaced by $P^{(2s-1)}$. Figure 4.21 shows that using ϑ_s as the superframe send time for each session s, over 97% of the packets of each session reach the sink node with a latency less than or equal to 20 time slots (a network designer should calculate and plot $p_{\text{net}}^{(t_d)}$ using all possible superframe send times, but only one plot is displayed here). We will set the packet TTL counter $t_{\text{TTL}} = 30$ time slots to have a large margin over the 20 time slots needed for over 97% connectivity.

We would like each camera node in the network to process 5 frames per second, which

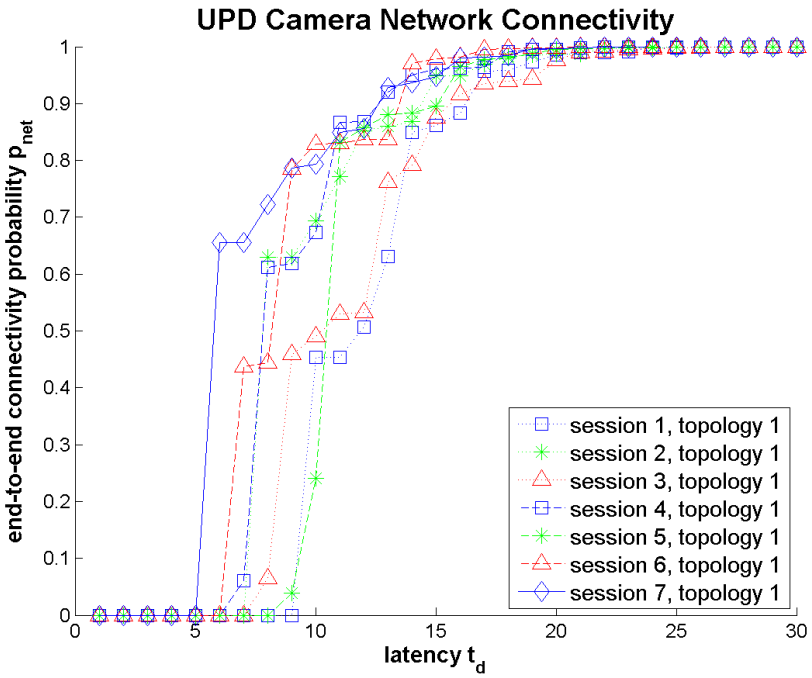

Figure 4.21. Calculated end-to-end connectivity for UPD on the camera network routing topology of Figure 4.19. Links have probability 0.9 or 0.6, as explained in the text of Section 4.5.1. The superframe send time is $\vartheta_s = 2s - 1$ for these plots.

suggests that if there are 100 time slots per second (e.g., WirelessHART) each session will generate and inject a new packet into the network every 20 time slots, i.e., $\zeta = 20$. Therefore, the superframe send times for each session are all the odd time slots,

$$\begin{aligned} \mathcal{F}_s &= \Big\{ \big((\vartheta_s - 1) + k \cdot 20 \pmod{14} \big) + 1, \quad \forall k = 1, \ldots, 13 \Big\} \\ &= \{1, 3, 5, \ldots, 13\} \qquad \text{(since } \vartheta_s \text{ is odd for all sessions)} \quad . \end{aligned} \tag{4.38}$$

Hereafter, when we refer to "all superframe send times" we mean $\mathcal{F}_s$ calculated from (4.37), like (4.38), and *not* all the slots in the superframe.

The UPDMC model also allows us to calculate the traffic distribution in the network. Figure 4.22 shows the probabilities $\tilde{\boldsymbol{p}}^{(t)}$ that a packet from session 1 visits the nodes in the network before its TTL counter expires. Note from the figure on the top left that if all the packets for the session have the same superframe send time, the traffic distribution on the network is very unbalanced, especially since we have high link probabilities in the network. Packets will tend to choose the same path through the network. The figure on the top right assumes $\mathcal{F}_s$ is given by (4.38) and shows that the traffic distribution can be

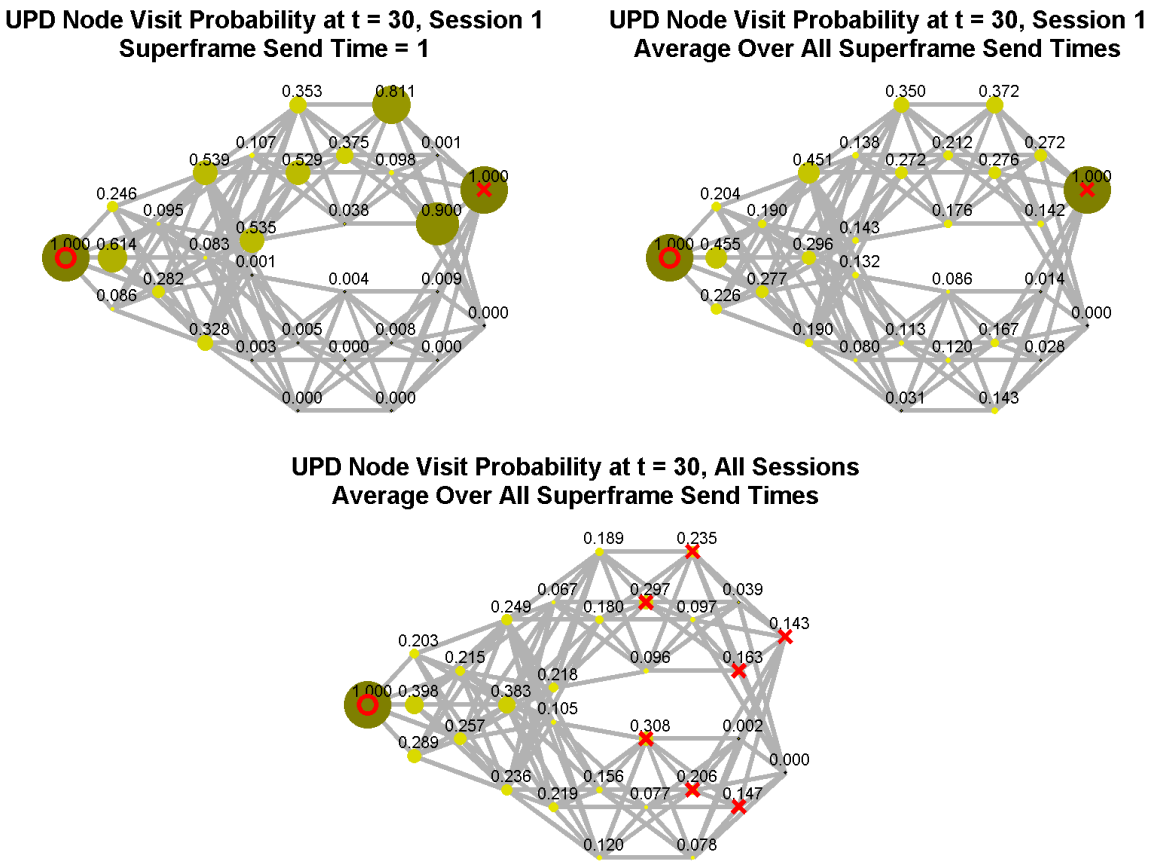

Figure 4.22. Node Visit Probability for UPD on the camera network routing topology of Figure 4.19. The size and darkness of the circles superimposed on the nodes correspond to the probability of visiting the node (also labeled next to the node). 'X' denotes source(s), 'O' denotes sink(s).

much better when each session s has multiple superframe send times. The bottom figure shows the traffic distribution averaged over all 7 sessions, where the traffic distribution of each session s in turn is averaged over all superframe send times $\mathcal{F}_s$.

Next, we use the UPDMC model to calculate the expected Tx / Rx energy usage (used synonymously with "energy usage" below) on the nodes in the network when we route a single UPD packet. The Tx / Rx energy usage per packet is calculated by weighting the expected Tx / Rx count to route the packet by the energy usage per transmission or reception. The top plot in Figure 4.23 shows the energy usage of each node when we route a packet for session 1. Sending all packets with the same superframe send time results in unbalanced energy usage throughout the nodes in the network. However, using all the

superframe send times in $\mathcal{F}_1$ gives more balanced energy usage, with all nodes using on average less than 767 μJ per packet.

The bottom plot of Figure 4.23 shows that even though each of sessions 2–7 uses many superframe send times (all the odd time slots), many nodes do not use any energy because they do not relay the packet. Only a subset of the nodes in the network are on a routing path from each of these sessions' source nodes to the sink. For instance, sessions 6 and 7 route packets through less than half of the nodes because the other nodes are either in another hallway across the middle of the building or have a higher minimum hop count to the sink. After averaging the energy usage over all the sessions, the energy usage across the network is more balanced. Node 1, the sink node, has the highest energy usage, but still uses less than 767 μJ of energy per packet. This is close to the theoretical minimum of 690 μJ, which assumes that the sink node only turns on the receiver once for each packet.

To verify that queuing does not impact the packet latency too much, we simulated the network for 6000 time slots. Recall that $\zeta = 20$ so we are simulating 300 packets per session and that $\mathcal{F}_s$ for all the sessions are the same and is the set of all the odd time slots. Figure 4.24 shows that for all sessions, fewer than 1% of the packets were dropped, given $t_{\text{TTL}} = 30$. In fact, for each session, over 75% of the packets (sometimes up to 88%) had a latency of less than 15 time slots. We conclude that the network can reliably support the camera nodes if each camera processes images at 5 frames per second.

Next, we use the DSFMC model to evaluate DSF and compare it with UPD on the camera network routing topology. In our example, the packet generation rate $1/\zeta$ for DSF must be lower than the packet generation rate for UPD. This is because DSF has multiple network packets, which takes up more network bandwidth. For instance, at the last stage before the sink, each node in the stage may contain a network packet, so they all must be scheduled to transmit to the sink before the corresponding session packet can be cleared from the network. In fact, since each node is scheduled to transmit once per superframe, all the sessions *together* cannot generate and inject packets into the network faster than rate

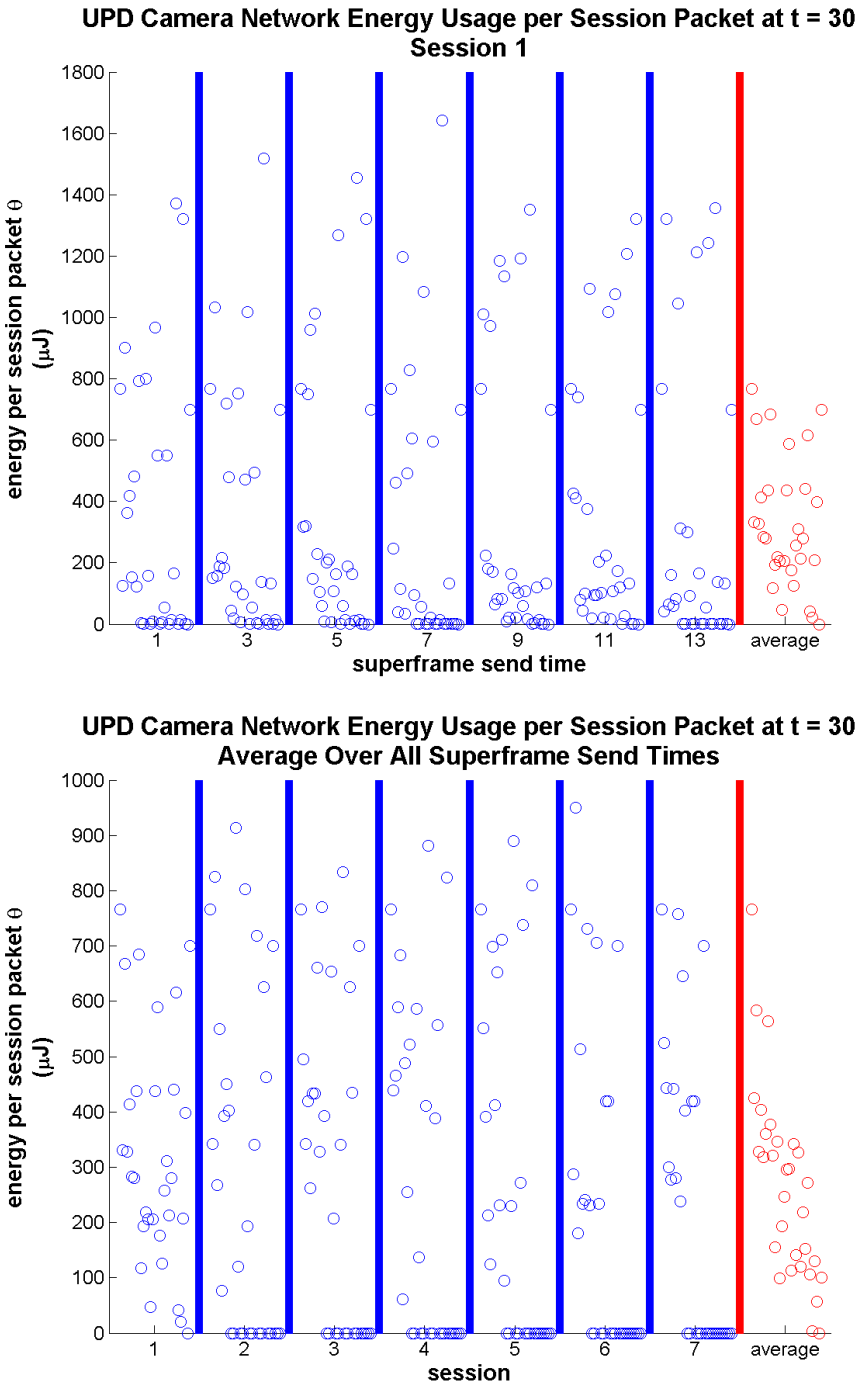

Figure 4.23. Raster plot of the calculated Tx / Rx energy usage for UPD on the camera network routing topology of Figure 4.19. Each point in the raster plot (marked by a circle) represents the expected Tx / Rx energy usage for a node, with points ordered from left to right by ascending node ids. (top) Each column represents one of session 1's possible superframe send times. The last column plots the energy usage of each node averaged over all of session 1's possible superframe send times. (bottom) Each session's column plots the energy usage average over all of that session's possible superframe send times. The last column plots the energy usage of each node averaged over all the sessions.

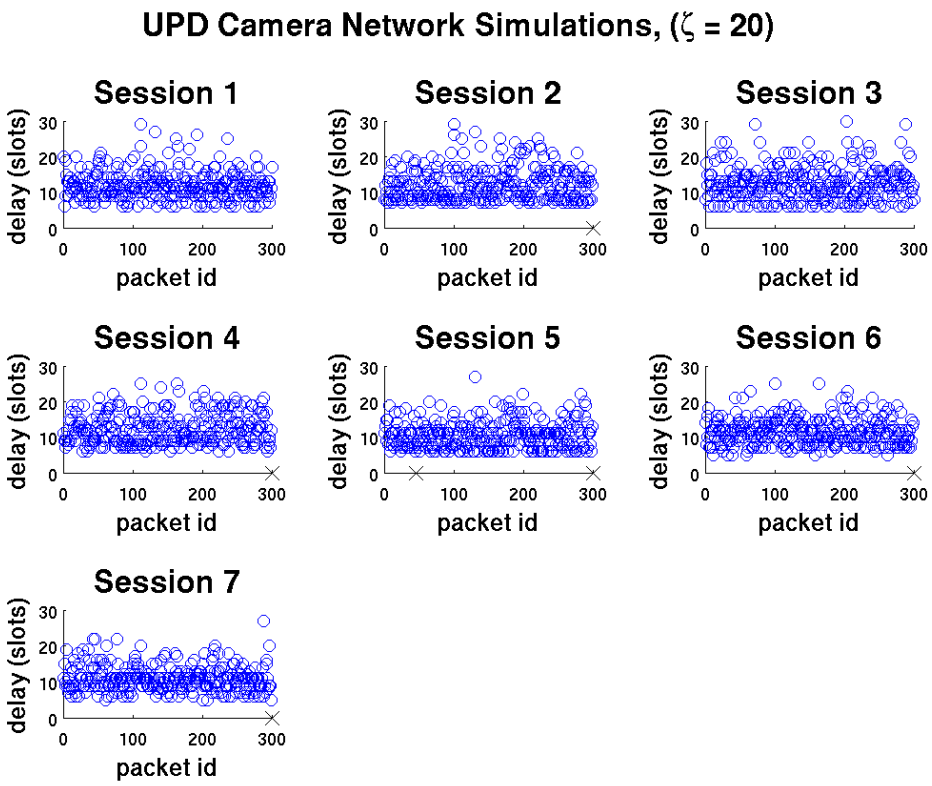

Figure 4.24. Raster plot of each sessions' packet latencies for UPD on the camera network routing topology of Figure 4.19. A black 'X' on the x-axis represents a packet drop. A dropped packet for the last simulation run (largest packet id) may be a result of the simulation time running out, i.e., the packet's TTL counter has not yet expired.

$1/T$, i.e.,

$$\sum_{s=1}^{S} \zeta_s = S \cdot \zeta \quad \leq \quad T$$

where $S = 7$ and $T = 11$ in our scenario. Thus, the DSF schedule described in Section 4.5.1 cannot support the packet generation rate $\zeta = 1/20$ used in our UPD simulation above. Figure 4.25 shows a high packet drop rate when we run DSF with $\zeta = 1/20$ — sessions 2, 3, 4, and 5 drop $> 92\%$ of their packets. The packets queue in the network and then their TTL counter expire (assuming $t_{\text{TTL}} = 42$, as will be explained below).

If $\zeta \geq 77$ time slots, then the queues on the nodes in the network will be bounded even if we do not drop old packets from the network. We set $\zeta = 100$ time slots in our simulations, which corresponds to each session (camera node) generating 1 packet per second if the network has 100 time slots per second. To reduce packet latencies, we space apart

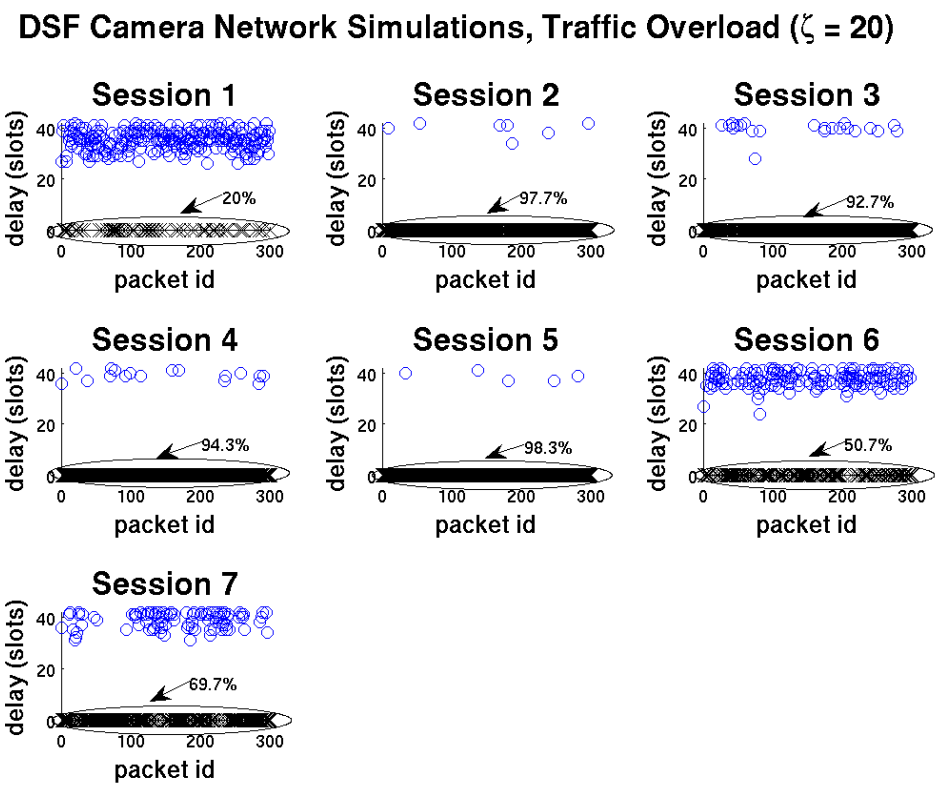

Figure 4.25. Raster plot of each sessions' packet latencies for DSF on the camera network routing topology of Figure 4.19 under an excessively high traffic load (each session generates a new packet every 20 time slots). A black 'X' on the x-axis represents a packet drop. A dropped packet for the last simulation run (largest packet id) may be a result of the simulation time running out, i.e., the packet's TTL counter has not yet expired.

the session send times for each session by at least 11 time slots to reduce queuing in the network. We set $\boldsymbol{\vartheta} = [1\ 16\ 26\ 40\ 51\ 66\ 77]$ so that in the first 100 time slots each session generates a packet when its source node is scheduled to transmit.

As mentioned in Section 4.4, because DSF does not have retransmissions, packets are delivered to the sink within a bounded time window. Therefore, we will set the TTL of each packet to the maximum packet latency, which can be determined from the schedule as shown in Figure 4.17. In our example, the superframe send times are

$$\mathcal{F}_s = \Big\{\big((\vartheta_s - 1) + k \cdot 100 \pmod{11}\big) + 1, \quad k = 0, 1, \ldots, 10\Big\} = \{1, \ldots, 11\}$$

which is depicted as the light gray boxes in Figure 4.17. Therefore, the maximum packet wait time is 10 time slots. We can calculate from the schedule that the minimum and

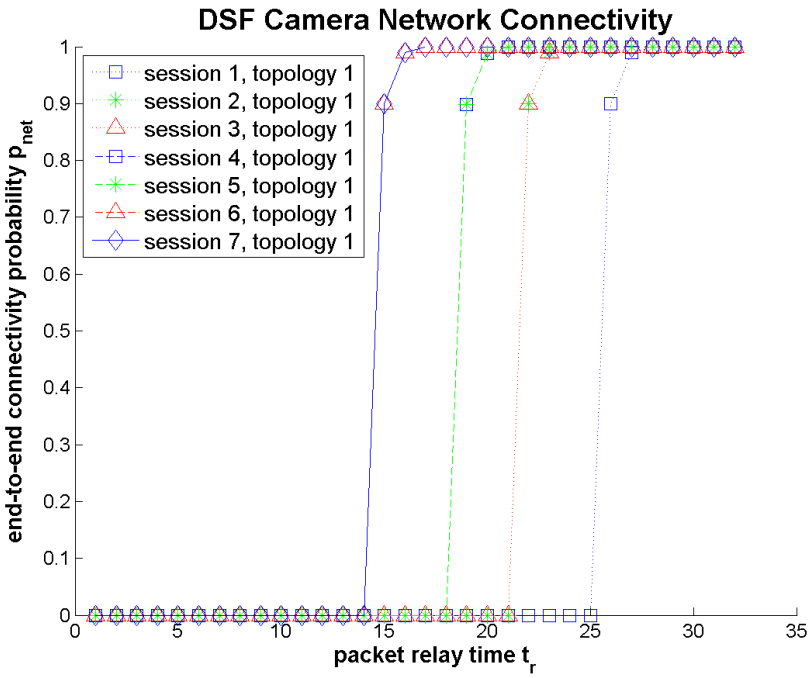

Figure 4.26. Calculated end-to-end connectivity for DSF on the camera network routing topology of Figure 4.19. Links have probability 0.9 or 0.6, as explained in the text of Section 4.5.1. The x-axis of the plot is the packet relay time, as explained in Figure 4.17.

maximum packet relay time for session 1 is 26 and 32 respectively, so the maximum packet latency is 42 time slots for session 1. It turns out that the maximum packet latency for session 1 is greater than or equal to the maximum packet latency for the other sessions, so we set $t_{\text{TTL}} = 42$.

Figure 4.26 shows the end-to-end connectivity of DSF as a function of the packet relay time t_r.[12] The values of $p_{\text{net}}^{(t_{\text{TTL}})}$ for all the sessions are all greater than 99.9981%.

Figure 4.27 shows the traffic distribution under DSF for session 1 and the average traffic distribution over all sessions. Note that none of the nodes have a packet visit probability of 1, but some visit probabilities are displayed as 1 due to roundoff errors. Unlike UPD, the traffic distribution in DSF is not dependent on the packet's session send time. The DSF network traffic is more even than the UPD network traffic.

[12] We chose not to plot $p_{\text{net}}^{(t_d)}$ (using t_d on the x-axis) like in Figure 4.21 because the graphs for each session would look like they overlap if we use the same superframe send times. For example, if we set $\mathcal{F}_s = \{1\}$ for all the sessions, then for any t_d, $p_{\text{net}}^{(t_d)}$ for any pair of sessions differ by less than 0.002. If instead we use ϑ as the superframe send time, the plot would be too wide.

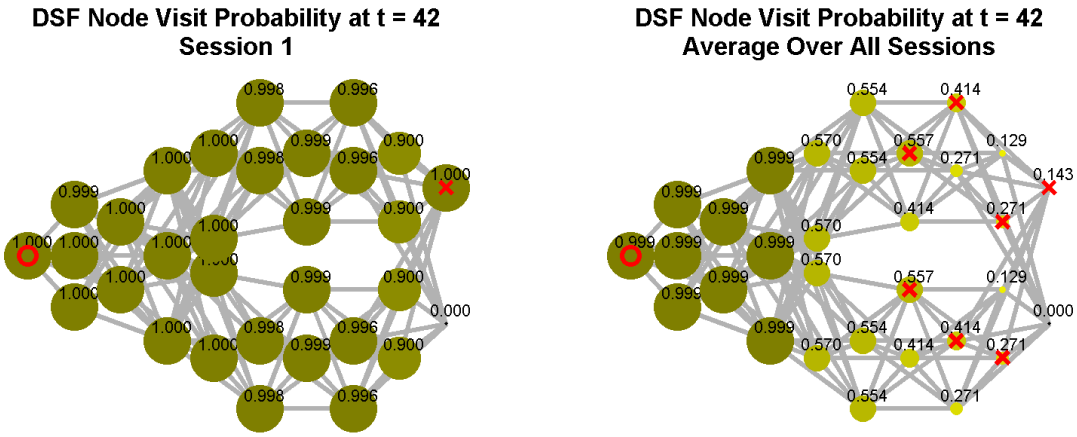

Figure 4.27. Probability that a packet visits a node at or before time t for DSF on the camera network routing topology of Figure 4.19. The size and darkness of the circles superimposed on the nodes correspond to the probability of visiting the node (also labeled next to the node). 'X' denotes source(s), 'O' denotes sink(s). Note that none of the nodes have a visit probability equal to 1, but some probabilities are displayed as 1 due to roundoff errors.

Figure 4.28 shows the energy usage of the nodes when transmitting a single packet for all the sessions.[13] Naturally, the nodes with the most number of incoming links (nodes 7, 8, 9) have the highest Tx / Rx energy usage.

To verify that we properly designed our schedule to reduce the effects of queuing in the network, we simulated the network and plotted the results in Figure 4.29. None of the packets are dropped, and in fact over 78% of the packets for sessions 2 – 7 have latencies $t_d \leq 30$ time slots. Note that the packet latencies for each session vary in a periodic fashion over the packet ids. This is because the superframe send times for session packets vary in a periodic fashion over packet ids. The range of packet latencies for each session is due to the range of possible packet wait times t_w for each session packet.

In our choice of schedules for DSF and UPD, DSF has much lower network bandwidth than UPD because multiple copies of a packet are present in the network. To utilize the bandwidth more efficiently in DSF, we would need to modify the schedule and use less nodes per stage to relay packets, at a cost of lower reliability. Also, Figures 4.21 and 4.26 show that UPD tends to have lower packet latencies. Figures 4.23 and 4.28 show that UPD also tends to have lower energy usage per session packet. However, Figures 4.22 and 4.27 show

[13] We didn't need to use (4.33) in this scenario because it was easy to directly count the number of transmissions and receptions given the traffic distribution.

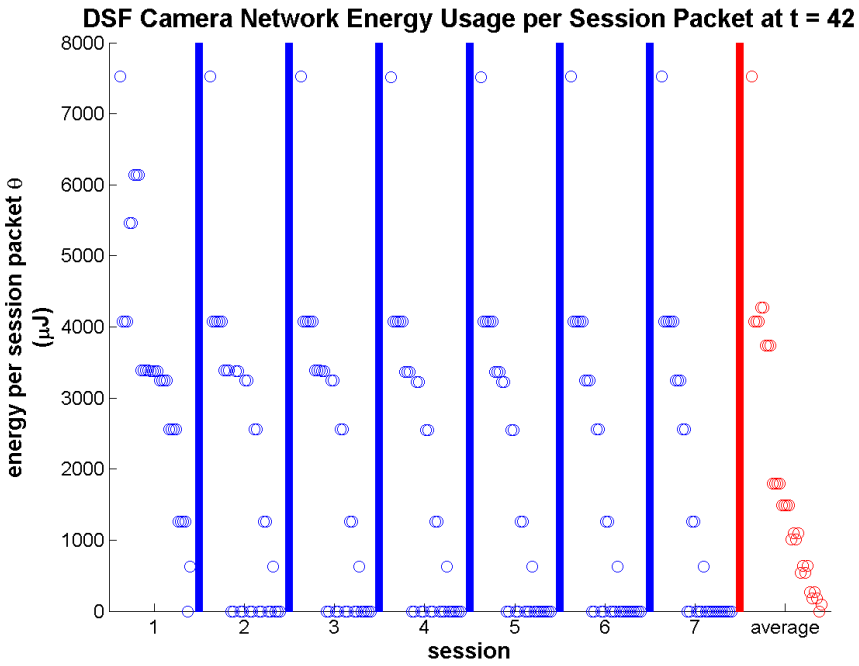

Figure 4.28. Raster plot of the calculated Tx / Rx energy usage for all sessions with DSF on the camera network routing topology of Figure 4.19. Each point in the raster plot (marked by a circle) represents the expected Tx / Rx energy usage for a node, with points ordered from left to right by ascending node ids. Each session's column plots the energy usage average over all of that session's possible superframe send times. The last column plots the energy usage of each node averaged over all the sessions.

that DSF has more even traffic distribution and Figures 4.24 and 4.29 show that DSF tends to have more predictable packet latencies when packets are injected into the network at regular intervals. After using the modeling tools and simulations presented in this section, a network designer would likely choose our UPD schedule over our DSF schedule for this network.

Note that we have simplified this case study by only considering 7 sessions. To fully evaluate the protocols, routing topologies, and schedules for our camera network scenario, we need to consider how all 31 sessions (one per camera) may interact. For instance, we need to stagger the superframe send times of the sessions more carefully if we want to avoid queuing in the network. We may still be able to take advantage of the assumption that only certain cameras can view a target simultaneously, so that a single target cannot trigger all the cameras to send packets back to the wired node. Therefore, some pairs of sessions will never simultaneously have packets to send.

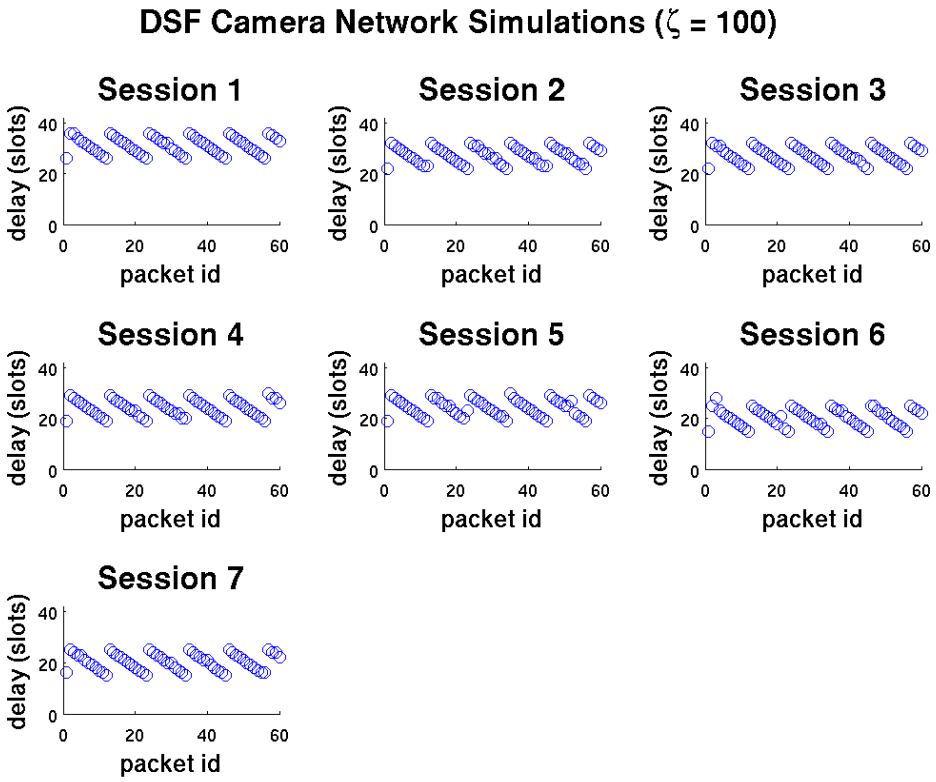

Figure 4.29. Raster plot of each sessions' packet latencies for DSF on the camera network routing topology of Figure 4.19, where each session generates a new packet every 100 time slots. No packets were dropped in these simulations.

4.5.2 Case Study 2: Real-Time Control System

The control applications that can benefit from integration with the wireless sensor networks modeled in this dissertation are ones where the plant (system to be controlled) has interacting dynamics physically distributed over a large area that can be monitored by sensors and controlled by actuators from fixed locations. The reason the sensors and actuators need to be at fixed locations is because the WSNs studied here have a static network topology. An exception, where the actuators can move, is a particular setup of pursuit-evasion games between robots [76]. Here, a WSN measures and estimates the position of evader robots and relays it to a basestation, which then computes the evaders' trajectories and uses a reliable, high-power, wireless link to send this information to a group of pursuer robots.

Motivation to Study Chemical Processing

Chemical process control is a large class of applications that can benefit from WSNs. A high-level survey of the literature [4; 5] reveals that the steps for creating chemical products often involve multiple interacting process loops across the plant, not just a simple linear progression from raw materials to final product. For instance, in the pulp and paper industry the chemicals used to break down wood chips are recovered from the byproducts through a multi-step chemical recovery process before they are fed back into the wood chip digester (See Figure 4.30) [73].

Control across a large plant is often coordinated in a hierarchical fashion, with supervisory controllers issuing setpoints across the plant to local control loops. Model Predictive Control (MPC) is popular for supervisory control while PID Control is popular for local unit control. A multi-hop WSN would integrate well with supervisory control because it can span the coverage of a factory. Mercangöz and Doyle noted in [73] that a paper machine alone can be up to 200 m in length and 10 m in height, well within the range of a multihop 802.15.4 network. Furthermore, supervisory control can operate at a slower time scale than local control, meaning that a WSN node can transmit less frequently and sleep for longer periods of time, prolonging network lifetime.

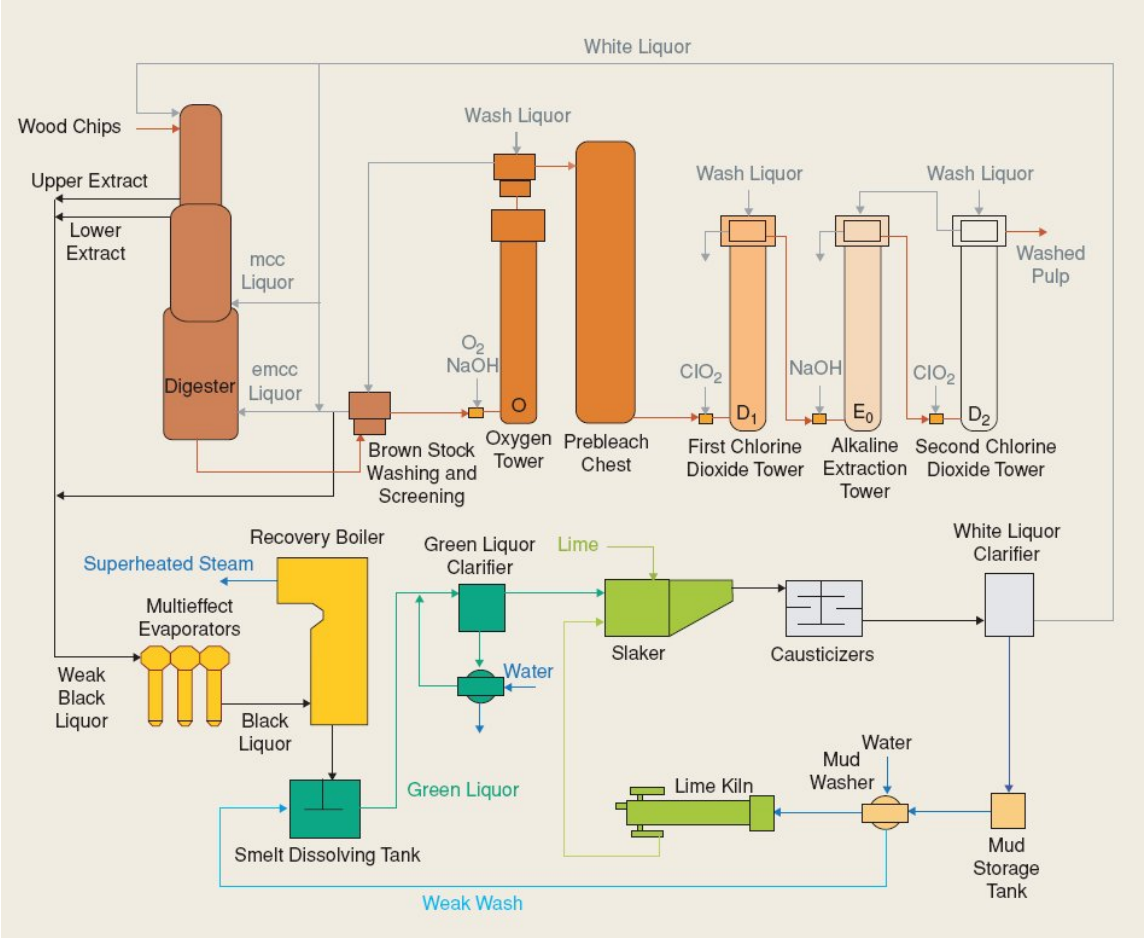

Figure 4.30. Pulp mill to process wood chips into washed pulp for making paper. The "white liquor" (NaOH and Na_2S) is recovered from the "black liquor" coming out of the wood chip digester through a series of chemical recovery loops. (Figure courtesy of [73].)

The benefits of wireless control in a factory include lower cost of installation, no repair of worn cables, and safety from not having cables [91]. In fact, technology is being developed to harness waste heat from industrial processes to power WSN nodes such that the batteries will not have to be replaced, saving even more on maintenance costs [91]. The potential to add extra sensors and actuators at lower cost in more locations can yield more fine-grained control of the plant. In the future, wireless sensors and actuators may also make factories easier to reconfigure to make different types or grades of products because wireless components are easier to reposition on the production line.

Better plant-wide control of chemical processing may yield a bigger return on investment than other types of industrial automation such as car assembly because of the variability

in the inputs and the volume of the production. Many chemical processing plants such as oil refineries and pulp and paper mills consume raw natural resources of varying grades but need to produce products with specific grades. Also, poorer plant-wide control of the processing units means that larger intermediate surge tanks are needed between the units to isolate the dynamics of the different processes. Mercangöz and Doyle point out in [73] that pulp and paper mills can have a production capacity on the order of 10^5 tons per year and that surge tanks in these plants can increase capital costs and inventories while reducing the agility to switch between different paper product grades. They also point out that a 600,000-ton annual capacity pulp and paper mill can consume up to 300,000 kg/h of process steam and over 40 MW of electricity, so there is significant room for energy savings by using better control.

For these reasons, our case study of real-time wireless networked systems will be on a chemical processing control system running over a WSN. Specifically, we'll be looking at a subsection of a pulp and paper mill. Typically, the sampling times for local control loops are on the order of seconds to tens of seconds and the sampling times for supervisory control are on the order of tens of seconds to minutes [13]. At these time scales, even CSMA media access with low-power listening [81] can meet the latency requirements of a single control loop. However, the traffic load would be greater if the plant has multiple control loops and TDMA scheduling would utilize bandwidth more efficiently. Besides, frequency hopping is often necessary in a cluttered factory environment to avoid multi-path fading [96], and is not compatible with CSMA on 802.15.4 radios because these radios can only listen to one radio channel at a time. The TDMA mesh networking protocols modeled in this dissertation are a good fit for wireless control in chemical processing plants.

Pulp Mill Plant Model

Large models of industrial chemical processing plants [25; 86] and even wastewater treatment plants [46] are available on the web as benchmarks to test control strategies. However, since this is not a dissertation on chemical process control, we adopt a simple model provided by the Model Predictive Control Toolbox in MATLAB® for our case study.

The case study is titled "MPC Supervisory Control of a Two Stage Thermo-Mechanical Pulping Process" and is documented in [3; 106].

Figure 4.31 shows the process by which pulp is diluted and pressed to the proper consistency (defined as the ratio of dry mass flow rate to overall mass flow rate) for making newsprint further down the production line. The goal is to regulate the pulp consistency and reduce energy costs while meeting operational constraints. These constraints include:

- keeping the power usage on each refiner under maximum rated values,
- keeping the vibration level on the refiners under a threshold to prevent the refiner plates from clashing,
- keeping the pulp consistency within a tolerance band to prevent fiber damage and the blow line from plugging up, and
- keeping the inputs within physical limits.

There are five inputs and six outputs to this plant. The inputs / manipulated variables are the setpoints for the two gap controllers regulating the distance between the refiner plates, the dilution flow rates to the two refiners, and the rpm of the pulp screw feeder. The outputs are the motor loads, output pulp consistency, and refiner vibration of the two refiners. The details of the plant dynamics are described by the Simulink® diagram in [106].

Model Predictive Controller

A discrete time model predictive controller uses an internal model of the plant and the measured outputs from the plant to predict the behavior of the plant during a finite time *prediction horizon* T_p. It simultaneously computes the optimal control over a finite time *control horizon* T_c. The MPC controller then applies the computed control input for the first time step to the plant and discards the remaining control inputs. At the next time step, a new measurement is taken from the plant and this process of computing the next

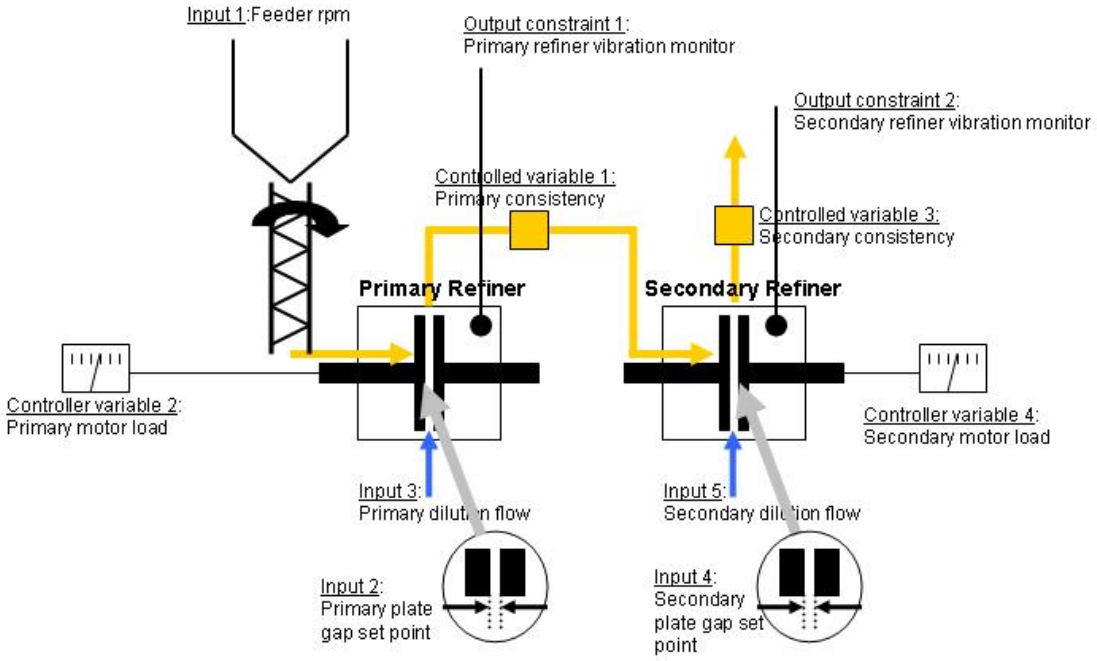

Figure 4.31. MPC Supervisory Control of a Two Stage Thermo-Mechanical Pulping Process. See text for details. (Figure courtesy of [106].)

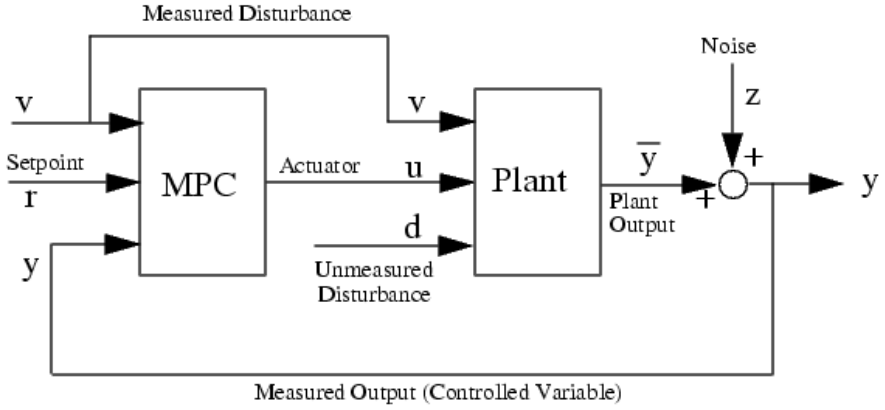

Figure 4.32. General setup of a model predictive control (MPC) control loop in MATLAB. See text for details. (Figure courtesy of [106].)

control input is repeated. The general setup of a MPC control loop for a SISO (single input single output) plant is shown in Figure 4.32.

Our problem is MIMO (multiple input multiple output) and does not have any measured

disturbances $\boldsymbol{v}$ or unmeasured disturbances $\boldsymbol{d}$ in the model maintained by the controller, although there is white noise in the model of the plant used in the simulations. Let $\boldsymbol{u} \in \mathbb{R}^5$ be the manipulated variable (control input) to the plant, $\boldsymbol{y} \in \mathbb{R}^6$ be the measured plant output, and $\boldsymbol{r} \in \mathbb{R}^6$ be the setpoint. The mapping of plant inputs and outputs to these vectors is given in Table 4.1. The system sampling time is 0.5 minutes, $T_p = 20\,\text{min}$, and $T_c = 5\,\text{min}$. After converting time to discrete time steps k, the optimization problem solved by the model predictive controller at each time step is

$$\begin{aligned}
\min_{\{\boldsymbol{u}(k),\ldots,\boldsymbol{u}(k+9)\}} & \sum_{\kappa=k}^{k+39} \left(\boldsymbol{y}(\kappa) - \boldsymbol{r}(\kappa)\right)^{\mathsf{T}} Q \left(\boldsymbol{y}(\kappa) - \boldsymbol{r}(\kappa)\right) + \\
& \sum_{\kappa=k}^{k+9} \left(\boldsymbol{u}(\kappa) - \boldsymbol{u}(\kappa-1)\right)^{\mathsf{T}} R \left(\boldsymbol{u}(\kappa) - \boldsymbol{u}(\kappa-1)\right) \\
\text{s.t.} \quad & \text{at all time steps} \\
& 0 \le u_1 \le 35 \qquad y_1 \le 1 \\
& 0 \le u_2 \le 1 \qquad y_2 \le 0.45 \\
& 70 \le u_3 \le 250 \qquad y_3 \le 1 \\
& 0 \le u_4 \le 1 \qquad y_4 \le 9 \\
& 70 \le u_5 \le 250 \qquad y_5 \le 0.4 \\
& \qquad\qquad\quad y_6 \le 9
\end{aligned} \tag{4.39}$$

where

$$Q = \text{diag}(0, 10, 0, 1, 10, 1) \qquad R = \text{diag}(0.1, 10, 0.1, 10, 0.1)$$

$$\boldsymbol{r} = \begin{bmatrix} 0 & 0.4 & 0 & 8.5 & 0.3 & 6 \end{bmatrix}^{\mathsf{T}} .$$

The MPC controller uses an internal model of the plant and the past observations to predict the future state of the plant $\boldsymbol{x}(k), \ldots, \boldsymbol{x}(k+39)$ and the future output $\boldsymbol{y}(k+1), \ldots, \boldsymbol{y}(k+39)$.

In this case study, we do not modify the MPC controller provided by the MATLAB MPC toolbox in any manner to account for the lossy, delayed wireless communication channel. Our goal is to evaluate how the lossy, delayed wireless communication channel impacts the performance of the control system.

Table 4.1. Mapping of inputs / outputs to variables in MPC optimization problem

variable	value
u_1	feeder (rpm)
u_2	primary gap set point
u_3	primary dilution flow set point (gallons per minute)
u_4	secondary gap set point
u_5	secondary dilution flow set point (gallons per minute)
y_1	primary motor vibration
y_2	primary output pulp consistency
y_3	secondary motor vibration
y_4	primary motor load (MegaWatts)
y_5	secondary output pulp consistency
y_6	secondary motor load (MegaWatts)

Wireless Network

The wireless network used to relay control and observation packets is depicted in Figure 4.33. We assume the sensors for the vibration, consistency, and motor load for the primary refiner are all wired to a single wireless sensor node, and similarly for the secondary refiner. There is also a single supervisory controller located at one end of the plant, 4 hops away from the pulp processing equipment. There are 5 sessions, numbered 1 to 5:

1. controller to feeder,
2. controller to primary refiner,
3. controller to secondary refiner,
4. primary refiner to controller, and
5. secondary refiner to controller.

Sessions 1, 2, and 3 use topologies 1, 2, and 3 respectively, while sessions 4 and 5 share the same routing topology (topology 4 described in the caption of Figure 4.33). All the links in our network have transmission success probability $p_l = 0.9$.

We will use UPD to send packets over this network. The schedule depicted in Figure 4.34 includes links from all the routing topologies and is shared by all the sessions.

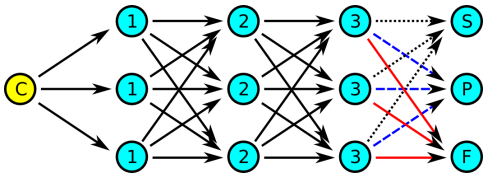

Figure 4.33. The 4 pulp mill network routing topologies. Topology 1 (solid black and solid red links) routes packets from the controller (C) to the feeder (F). Topology 2 (solid black and dashed blue links) routes from the controller to the primary refiner (P). Topology 3 (solid black and dotted black links) routes from the controller to the secondary refiner (S). Finally, topology 4 (all links depicted but with directions reversed) routes packets from the primary and secondary refiner to the controller. All links have success probability $p_l = 0.9$.

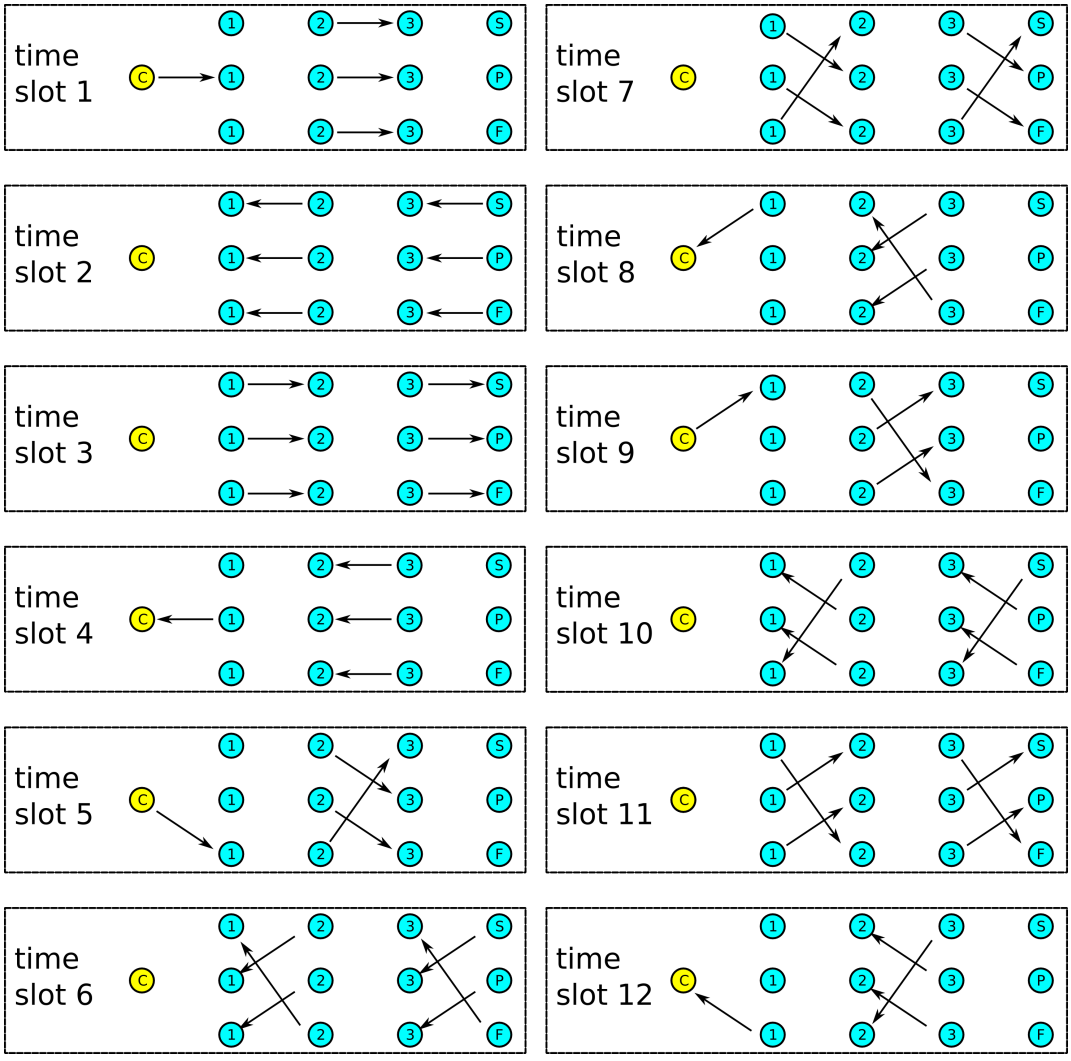

Figure 4.34. Schedule indicating active links for each time slot on pulp mill network. This schedule is shared by all the routing topologies. C stands for controller, F stands for feeder, P stands for primary refiner, and S stands for secondary refiner.

Let us assume that we have 10 ms time slots, like in WirelessHART [40]. The system sampling time is 30 sec, or 3000 time slots. If we wish to have all the nodes in the network operate with a 0.5% duty cycle (nodes sleep for 199 time slots and wake up 1 time slot to transmit), a node would wake up to send or receive a packet every 2 seconds. Let us assume the nodes are powered by AA batteries with 2500 mAh of charge (reasonable for alkaline batteries, [30]) and the current consumption is less than 25 mA when the mote is on and less than 10 μA when the mote is sleeping (reasonable for the TelosB platform, [21]). The sleep current is negligible given a 0.5% duty cycle, so the calculated expected lifetime of a node is approximately

$$\frac{(200)(2500\,\text{mAh})}{(25\,\text{mA})(24\,\text{h/day})(365\,\text{day/year})} \approx 2.3\,\text{years} \tag{4.40}$$

if the node transmits on every time slot that it is awake. This means batteries can be replaced using a once-a-year maintenance schedule. Let the *network lifetime* be defined to be the expected time until the first node runs out of energy. In the next section, we will calculate the network lifetime assuming that nodes only generate or relay packets (no baseline energy usage from sensing or extra processing), nodes can sense the channel to see if they need to receive a packet and quickly go back to sleep, and nodes consume negligible energy when sleeping or when sensing the channel for packet reception. The network lifetime will be higher than (4.40) because each node will not be transmitting or receiving a packet on every time slot it is awake.

Furthermore, if we discard packets with an end-to-end delay of more than 2 system sampling times, then t_{TTL} for each packet should be set to 6000 time slots. At a 0.5% duty cycle this would be 30 time slots when the network is awake. In reality, the TTL counter on the packet should be set to 30 since we can only decrement counters when the nodes are not sleeping.

In this and following discussions, we will assume that the duty cycle of all the nodes in the network are synchronized, meaning all nodes sleep and wake up at the same time. This allows us to frame our discussion around *network-awake* time slots instead of actual time slots. Hereafter and unless otherwise noted, time is counted in network-awake time slots, where we assume the nodes have a 0.5% duty cycle and 1 network-awake time slot =

200 actual time slots. Thus, 1 system sampling time = 15 time slots (network-awake time slots), meaning each session generates a new packet every $\zeta = 15$ time slots.

Calculations and Simulations

To check whether the choices for the duty-cycle and initial packet TTL counter discussed in the previous section are reasonable, we used the UPDMC model to calculate the end-to-end connectivity as a function of latency, $p_{\text{net}}^{(t_d)}$, and plotted it in Figure 4.35. As noted in the previous section, time is counted in network-awake time slots, meaning we do not count time slots when the network is sleeping. Again, we staggered the superframe send times of the sessions to reduce the queuing in the network. Using $\boldsymbol{\vartheta} = [1\ 5\ 9\ 2\ 6]$ so the first session packet is generated when the session's source is scheduled to transmit and

$$\mathcal{F}_s = \Big\{ \big((\vartheta_s - 1) + k \cdot 15 \pmod{12} \big) + 1, \quad k = 1, \ldots, 29 \Big\}$$

we get

$$\mathcal{F}_1 = \{1, 4, 7, 10\}, \quad \mathcal{F}_2 = \{2, 5, 8, 11\}, \quad \mathcal{F}_3 = \{3, 6, 9, 12\},$$
$$\mathcal{F}_4 = \{2, 5, 8, 11\}, \quad \mathcal{F}_5 = \{3, 6, 9, 12\} \quad .$$

Figure 4.35 uses $\boldsymbol{\vartheta}$ as the superframe send times. The figure shows that all the sessions have 97.96% of the packets reach the sink under 30 network-awake time slots, i.e., a delay under 2 system sampling times (assumes no queuing).

Next, we used the UPDMC model to calculate the traffic distribution on the network, shown in Figure 4.36. Notice on the left picture that a lot of traffic passes through node 7, the node to the left of the node representing the feeder in Figure 4.33. In fact, the right picture shows that much of the traffic of session 1 passes through node 7 despite using all the different superframe send times in $\mathcal{F}_1$. We cannot assume traffic will always be evenly spread throughout the network just because packets are injected into the network at different superframe send times.

Figure 4.37 plots the calculated energy usage using the UPDMC model and confirms that node 7 consumes the most energy from relaying the most traffic. In fact, sessions 1, 2,

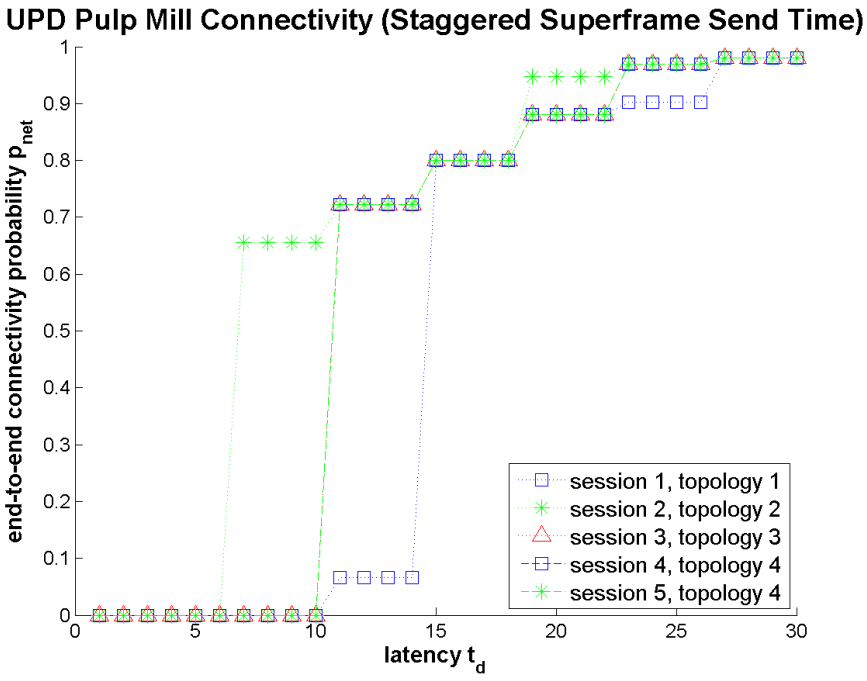

Figure 4.35. Calculated end-to-end connectivity as a function of delay for the 5 sessions on the pulp mill network. Assumes sessions 1 to 5 each have a superframe send time of 1, 5, 9, 2, and 6 respectively. Note that the plots for sessions 3, 4, and 5 lie on top of each other.

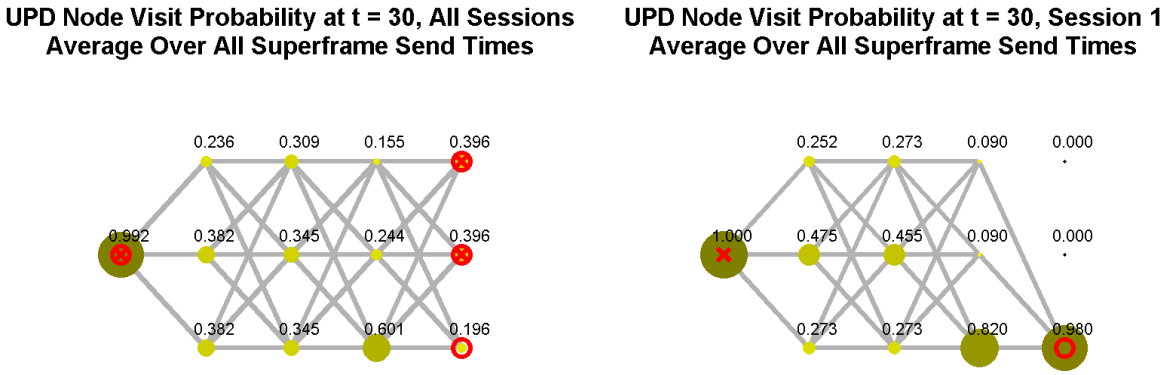

Figure 4.36. Probability that a packet visits a node at or before time t for UPD on the pulp mill network routing topology of Figure 4.33. The size and darkness of the circles superimposed on the nodes correspond to the probability of visiting the node (also labeled next to the node). 'X' denotes source(s), 'O' denotes sink(s).

and 3 all relay a lot of traffic through node 7. Thus, the network lifetime should equal node 7's lifetime. However, at our low packet generation rate of 1 session packet every 3000 time slots, the Tx / Rx energy usage has negligible impact on the node lifetime. Let us make the same assumptions for calculating a node's expected lifetime earlier: 2500 mAh batteries, the

nodes only relay packets (no baseline energy usage from sensing or processing), the nodes consume negligible energy while sleeping, and the nodes consume negligible energy when they wake up to sense the channel for receiving packets and quickly go back to sleep. The formula to convert from energy usage to lifetime is

$$\text{Node } i\text{'s lifetime (in years)} = \frac{(3\,\text{V})(2.5\,\text{Ah})(200 \cdot \zeta\,\text{time slots per session packet})}{(\theta_i^{(t_{\text{TTL}})}\,\text{J per session packet})(24\,\text{h/day})(365\,\text{day/year})(S\,\text{sessions})} \tag{4.41}$$

Using this formula, the expected network lifetime (node 7's lifetime) is ≈ 552 years, which is much higher than expected (usually, a node's battery is expected to last less than 10 years). Clearly, the baseline energy usage is no longer negligible in scenarios with very low traffic, so the calculated lifetime will be wrong. However, a real chemical processing plant will likely have much more than 5 sessions, meaning more data traffic. In that scenario, the network lifetime calculations may more closely match reality. If a network designer wishes to improve the traffic distribution and lower the Tx / Rx energy usage of the nodes in the network, he would have to tinker with the schedule or the superframe send times of the sessions.

After using the UPDMC model to plan and evaluate our network, we verified the performance of the entire closed loop networked control system by simulating it over a simulation time interval of 1 hour (time in simulation model, not time to run simulations). This corresponds to 1800 network-awake time slots (360,000 actual 10 ms time slots). The results of the wireless network simulation is depicted in Figure 4.38. Very few packets are dropped, and in each of the sessions roughly 37-48% of the packets have a system sample time delay of 1 (≤ 15 time slots) and roughly 46-58% have a system sample time delay of 2.

Figure 4.39 plots the ability of the networked control system to track the reference setpoints for the motor loads and the final pulp consistency. The simulations show significantly worse tracking even with low packet drop rates and packet delays of 1 or 2 system sampling times. In fact, the plant violates the primary motor load constraint $y_4 \leq 9$ from the optimization problem described by (4.39), with the motor load peaking at ≈ 11.9 MW.[14]

[14]Both the ideal channel NCS and the wireless channel NCS briefly violate the constraint $y_5 \leq 0.4$. This is because the initial output of the plant is close to this operating constraint.

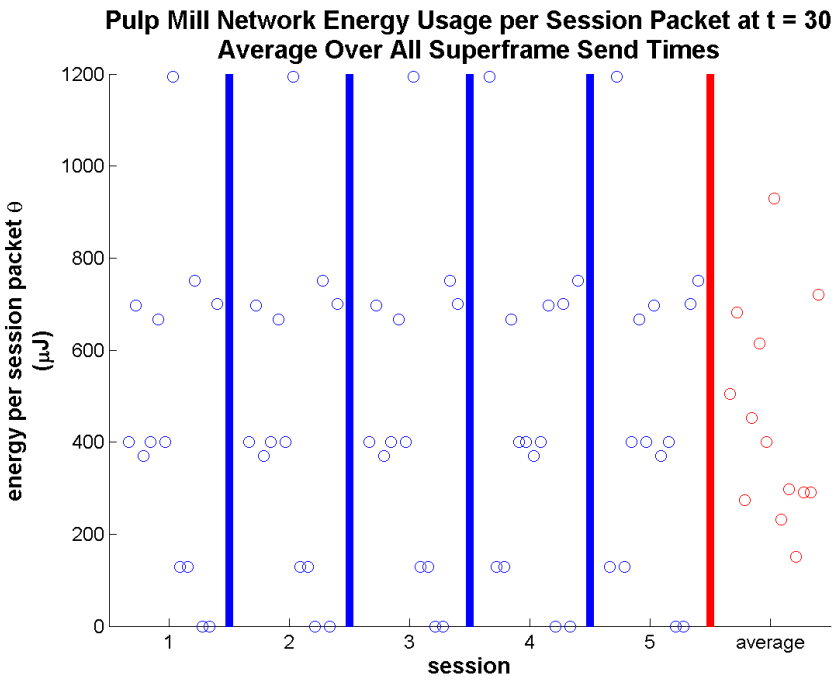

Figure 4.37. Raster plot of the calculated Tx / Rx energy usage for all sessions with UPD on the pulp mill network routing topology of Figure 4.33. Each point in the raster plot (marked by a circle) represents the expected Tx / Rx energy usage for a node, with points ordered from left to right by ascending node ids. Each session's column plots the energy usage average over all of that session's possible superframe send times. The last column plots the energy usage of each node averaged over all the sessions.

Table 4.2 compares the performance of the NCS using ideal communication channels (no packet loss or delay) with the performance using the lossy, delayed communication channels from our wireless mesh network. We compare the performance using the *Integral of Absolute Error* (IAE) and the *Integral of Time and Absolute Error* (ITAE) [62]. They can be computed as

$$IAE = \sum_{k=k_o}^{k_f} \big|\,\text{error}(k)\big| \tag{4.42}$$

$$ITAE = \sum_{k=k_o}^{k_f} t_k \cdot \big|\,\text{error}(k)\big| \tag{4.43}$$

where error is the difference between the output variable and the reference set point and t_k is the time in minutes at discrete time step k. Clearly, blindly substituting a wireless network, even with low packet loss rates and delays, can result in significantly worse system performance. This case study illustrates the magnitude of loss in efficiency (on the order of

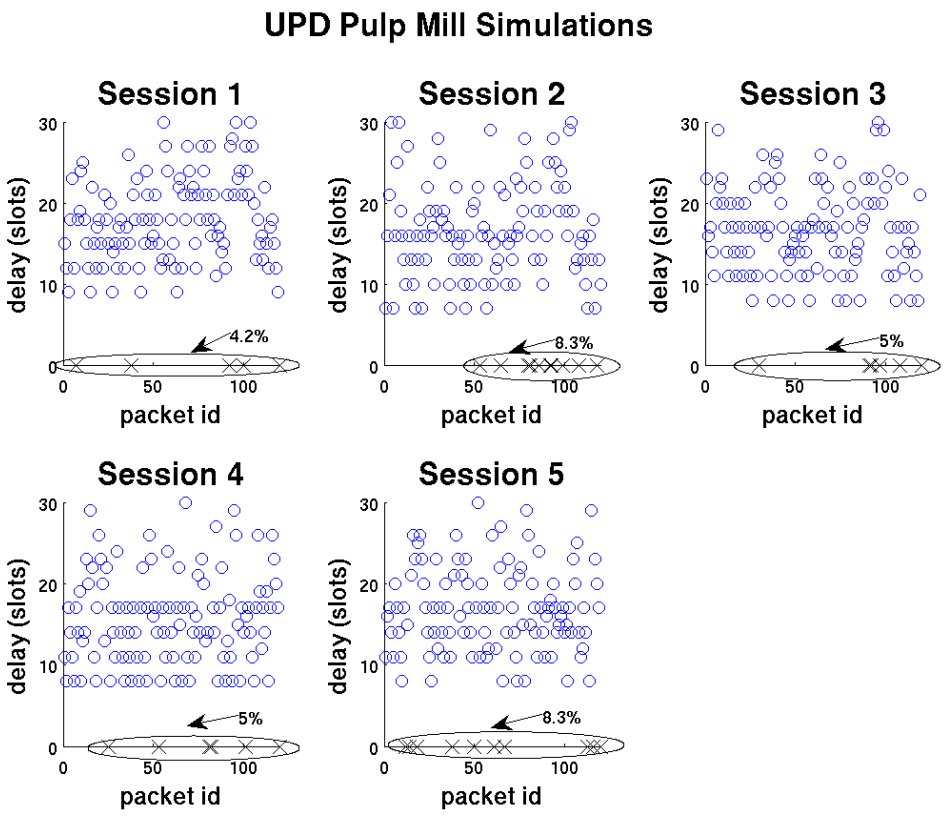

Figure 4.38. Raster plot of individual packet delays from simulations on the pulp mill network. A black 'X' on the x-axis represents a packet drop. A dropped packet for the last simulation run (largest packet id) may be a result of the simulation time running out, i.e., the packet's TTL counter has not yet expired. The plots are annotated with the percentage of packets that were dropped.

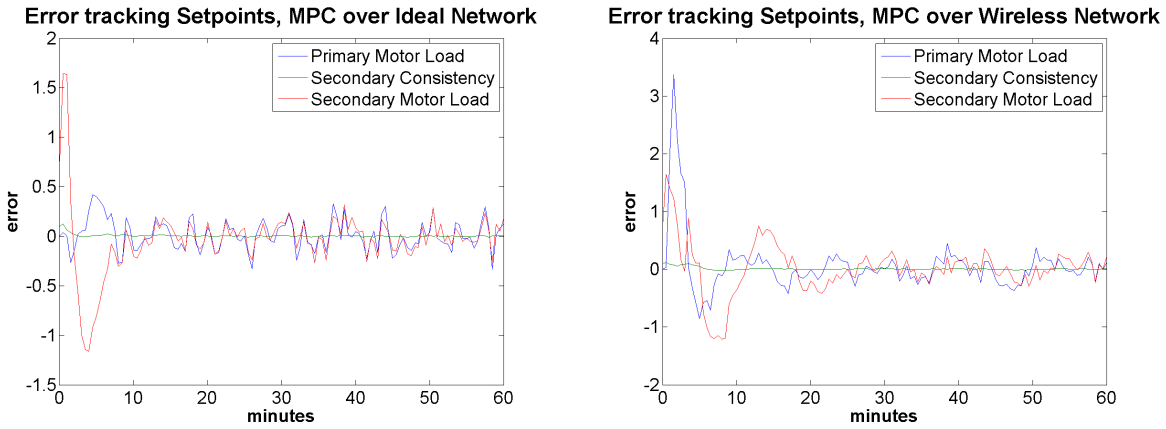

Figure 4.39. Difference between paper pulp mill output and reference setpoints under (left) ideal network conditions with no packet drops and no delay, and (right) a wireless network with the packet drop and delay statistics depicted in Figure 4.38.

Table 4.2. Pulp paper mill IAE and ITAE with and without packet losses + delay

output variable	Perfect Channel		Lossy, Delayed Channel	
	IAE	ITAE	IAE	ITAE
y_4, primary motor load (MegaWatts)	14.55	413.55	29.88	563.83
y_5, secondary output pulp consistency	0.96	17.89	1.68	25.57
y_6, secondary motor load (MegaWatts)	22.99	419.03	32.52	637.45

MegaWatts) and lower pulp consistency in a paper pulp mill. Fortunately, the performance of the wireless NCS gets better over time, but only after severely violating the primary motor load operational constraint.

Chapter 5

Conclusions and Future Work

Wireless Networked Systems will become more prevalent in the upcoming years as wireless sensor networks become integrated with real-time control and measurement systems to provide more points for sensing and actuation to interact with the environment. To manage the complexity of these systems, which may consist of hundreds or thousands of components connected by a lossy wireless communication medium, we need to develop a generic design methodology that can be applied across many similar systems. A key component of any design methodology is the definition of network abstractions to obtain network metrics, which in turn can be translated into measures of system performance using analysis tools such as the existing Networked Control System Theory.

5.1 Summary of Contributions

This dissertation provided a framework for analyzing the performance of real-time wireless networked systems. In particular, Chapter 4 develops the UPDMC and DSFMC models of two classes of networking protocols, UPD and DSF, given the routing topology and schedule. These models are used to compute the end-to-end connectivity metric, $p_{\text{net}}^{(t_d)}$, as an abstraction of the network for use in the design of the real-time measurement or control system. The UPDMC and DSFMC models also provide other useful metrics for charac-

terizing the network, such as the expected node energy consumption from relaying packets and the traffic distribution throughout the network.

Chapters 2 and 3 focus on the design of routing topologies and schedules that improve the reliability of packet delivery through the network. Directly optimizing the end-to-end connectivity metric $p_{\text{net}}^{(t_d)}$ by jointly designing the routing topology and schedule is difficult, so the routing topology and the schedule were designed separately. Both chapters studied how to increase the reliability of the network using path diversity, which is the use of multiple paths to route packets to the destination. Chapter 2 defined metrics to assess and compare mesh routing topologies and proposed a lightweight greedy algorithm to construct a routing topology from an undirected connectivity graph. Chapter 3 highlighted the difficulties in getting good path diversity with low latencies using repeating schedules and Greedy Maximal Matching scheduling. The APLM scheduling algorithm was proposed to generate the link matchings in the schedule at each time slot in a distributed manner. The mechanisms of the algorithm need improvement for general topologies but work well for layer-to-layer topologies.

5.2 Directions to Extend the Models and Algorithms

This section discusses the limitations of the models and algorithms presented in this dissertation, as well as how to extend or improve them.

Chapter 2 introduced the the robustness metric as a low computational complexity approximation of the path probability metric. Section 2.2.3 showed that there exist topologies where the difference between the robustness metric and the path probability is arbitrarily close to 1. A useful extension would be to derive a non-trivial bound (i.e., not 1) on the difference between the path probability metric and the robustness metric for a *given* routing topology G. It would also be useful to know what classes of DAGs have $r_{a\to b}$ close to $p_{a\to b}$.

Chapter 2 also introduced the robust minimum hop algorithm for constructing a routing topology that attempts to maximize, in a greedy fashion, a variant of the robustness metric from all nodes in the network to the sink. Section 2.4.3 pointed out that this may not

maximize the robustness metric between a *particular* node and the sink. Since there may be times when only be a few sources need to communicate to the sink, a useful extension is to modify the robust minimum hop algorithm to construct a routing topology that maximizes the robustness metric only between a pair of nodes. In fact, it would also be interesting to develop an algorithm that utilizes the rtFlow metric, or a variant of the metric, to construct a routing topology.

Chapter 3 demonstrated that short, repeating schedules often do not utilize all the paths in the network. It would be useful to develop a computationally tractable metric to measure path diversity. This will help us compare and generate good schedules. Preferably, this would be a metric that is well grounded in a link failure and routing model.

Chapter 3 also developed APLM, which used a simple 2-hop neighbor clustering heuristic to help partition the network and distribute the computation of the schedule. However, the throughput of the schedule was low because of the low number of links that were simultaneously scheduled in a time slot. In fact, the simple clustering technique throws away many links between nodes whose robust hop count differ by more than 1. Other clustering schemes should be evaluated, keeping in mind that larger clusters require more state and coordination between nodes. Ideally, one would be able to formalize the clustering and partitioning phase of a scheduling algorithm like APLM as an optimization problem that is negotiated by nodes in a distributed manner through "prices." If the clustering and partitioning phase is repeated over time to accommodate changes in the interference or topology, such a formulation would allow us to study the stability and rate of convergence of the scheduling algorithm as it adapts to the changing wireless conditions.

Scheduling also has a large affect on the queuing of packets on relay nodes in the network. In UPD, we want schedules that minimize the probability that packets queue at nodes in the *middle* of the network, since little can be done through scheduling to reduce queuing on nodes near the source and the sink. The source and sink will always be bottlenecks in the network. If a source is sending packets at a constant rate, a good direction for future work is to design a schedule that tends to spread *consecutive* packets along different paths through the network so that a link failure downstream does not back up the traffic.

While the effect of link estimation error on our UPDMC and DSFMC models is briefly covered in Chapter 4, this dissertation has not directly addressed how link estimation error affects the formation of routing topologies and schedules. A good direction for future research is how to formulate sensitivity to link estimation errors into the objectives of our routing topology and scheduling formation algorithms.

As pointed out in Section 4.4, the UPDMC model in Chapter 4 does not capture the benefits of having path diversity because the link transmission events are assumed to be independent from one time slot to the next. In reality, the quality of wireless links are often correlated over time because of short-term channel fading or interference [24]. If we have enough independent frequency channels between the nodes and the schedule is designed such that consecutive transmissions from a node to its downstream neighbor are on different frequency channels, the UPDMC model may be sufficient. However, it is worth exploring how we can extend the UPDMC model to better approximate the links in real wireless network deployment while remaining computationally tractable (i.e., the state space does not become too large). For instance, while exact modeling of the association / correlation between links is difficult, there may be a way to augment or combine the UPDMC model with the Gilbert-Elliot channel model for a point-to-point link such that the resulting model is a closer match to the conditions in a real wireless network deployment.

Finally, the UPDMC model only models the transmission of a single packet through the network. Thus, it implicitly assumes that consecutive packets are spaced far enough apart in time that they do not queue on relay nodes in the network. Another direction for further research is to incorporate queuing into our models, perhaps by borrowing results from Queuing Theory to apply to WSNs.

5.2.1 Flow Control and QoS

This dissertation focused on routing and scheduling, which are related to the network and data link layers of the OSI model. Another interesting area for research is to get a better understanding of when it makes sense to add flow control mechanisms, which are

part of the transport layer, to sensor networks. There is an extensive body of literature on modeling flow control in the Internet as a decentralized optimization problem (See [18] and references within). The objective is usually to maximize the total throughput while allowing for fairness amongst the users and low amounts of queuing in the network. Latency and reliability are usually not directly addressed in this formulation. Furthermore, flow control on the Internet is meant to address fluctuations in the demand for network bandwidth by the user, e.g., when the user wants to download a file. In this dissertation, we have only considered wireless networked systems that generate data packets at a fixed rate.

A slightly different direction for further study is to define metrics that measure the likelihood of packets queuing in the network given a set of independent flows (sessions) with different fixed packet generation rates. These metrics would allow a network designer to determine the maximum number of independent flows and data rates that can be supported by the network.

If a wireless network is shared by multiple applications, there may be a need to provide a *Quality of Service* (QoS) mechanism to prioritize traffic in the network. Such a mechanism would affect models of queuing and latency in the network. For instance, a simple QoS policy for a network supporting a real-time control system would be to discard older packets containing older measurements when the network is congested. Given newer packets, the older packets are less useful for estimating the state of the physical process we are measuring. Further study is needed on how to evaluate different QoS policies and relate it to the performance and stability of the entire wireless networked system.

5.2.2 Network Metrics for Adaptive Control

Thus far, the discussions in this dissertation revolved around using network metrics to adjust the network so it can better support the wireless networked system. When the wireless networked system is a real-time control system, it also makes sense to use the network metrics to tune the controller to varying network conditions. For instance, we can monitor the packet latency and success rate to estimate the network conditions and

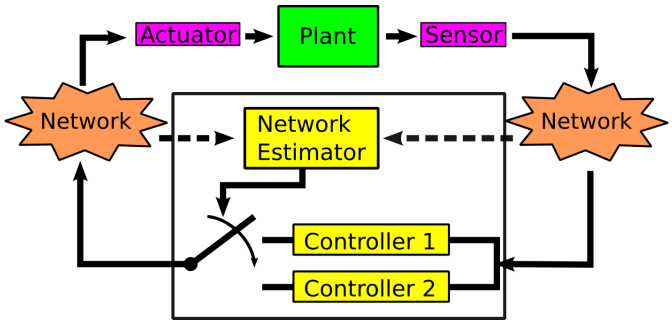

Figure 5.1. An example of a switching controller that adapts to varying network conditions. The Network Estimator is part of the full system controller. The dashed lines indicate the monitoring of network conditions, e.g., indirectly through checking timestamps and sequence numbers of delivered packets.

determine when to switch between controllers (See Figure 5.1), using a more aggressive controller when the network has a high packet delivery success rate with low latency. If we generalize this to switching among a "continuum" of controllers, this is effectively the same as designing a single controller which takes the network metrics as additional inputs. The challenge here is to develop the proper formulation to incorporate the network metrics into an optimal control framework.

Finally, it may be interesting to combine flow control with the design of controllers for NCSs over wireless sensor networks. A direction for further research is how to design a variable rate controller which adjusts the control commands and rate of sending these control commands to network conditions.

Bibliography

[1] Martin Becker, Ewoud Werkman, Michalis Anastasopoulos, and Thomas Kleinberger, "Approaching ambient intelligent home care systems," in *Pervasive Health Conference and Workshops*, 2006, pp. 1–10.

[2] Mihir Bellare, Oded Goldreich, and Madhu Sudan, "Free bits, pcps, and nonapproximability—towards tight results," *SIAM Journal on Computing*, vol. 27, no. 3, pp. 804–915, 1998. [Online]. Available: http://link.aip.org/link/?SMJ/27/804/1

[3] A. Bemporad, N.L. Ricker, and J.G. Owen, "Model predictive control - new tools for design and evaluation," *Proceedings of the American Control Conference (ACC)*, vol. 6, pp. 5622–5627, June-July 2004.

[4] B.W. Bequette, "Snapshots of process control," *IEEE Control Systems Magazine*, vol. 26, no. 4, pp. 28–29, Aug. 2006.

[5] B.W. Bequette, "Snapshots of process control II," *IEEE Control Systems Magazine*, vol. 26, no. 6, pp. 32–33, Dec. 2006.

[6] Dimitri P. Bertsekas and Robert Gallager, *Data Networks*. Englewood Cliffs, New Jersey: Prentice Hall, 1992.

[7] Dimitri P. Bertsekas and John N. Tsitsiklis, *Introduction to Probability*. Belmont, Massachusetts: Athena Scientific, 2002.

[8] Ramesh Bhandari, "Optimal physical diversity algorithms and survivable networks," *Second IEEE Symposium on Computers and Communications (ISCC)*, vol. 00, p. 433, 1997.

[9] Sanjit Biswas and Robert Morris, "Opportunistic routing in multi-hop wireless networks," *SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 69–74, 2004.

[10] A. Bonivento, C. Fischione, L. Necchi, F. Pianegiani, and A. Sangiovanni-Vincentelli, "System level design for clustered wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 3, no. 3, pp. 202–214, Aug. 2007.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2003.

[12] M. E. M. Campista, P. M. Esposito, I. M. Moraes, L. H. M. Costa, O. C. M. Duarte, D. G. Passos, C. V. N. de Albuquerque, D. C. M. Saade, and M. G. Rubinstein, "Routing metrics and protocols for wireless mesh networks," *IEEE Network*, vol. 22, no. 1, pp. 6–12, Jan.-Feb. 2008.

[13] J. J. Castro and F. J. Doyle, "A pulp mill benchmark problem for control: application of plantwide control design," *Journal of Process Control*, vol. 14, pp. 329–347, Apr 2004.

[14] A. Cerpa, J.L. Wong, L. Kuang, M. Potkonjak, and D. Estrin, "Statistical model of lossy links in wireless sensor networks," *Fourth International Symposium on Information Processing in Sensor Networks*, pp. 81–88, April 2005.

[15] Phoebus Chen and Shankar Sastry, "Latency and connectivity analysis tools for wireless mesh networks," in *Proceedings of the First International Conference on Robot Communication and Coordination (ROBOCOMM)*, October 2007.

[16] Phoebus Chen and Shankar Sastry, "Latency and connectivity analysis tools for wireless mesh networks," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2007-87, June 2007. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-87.html

[17] Phoebus Wei-Chih Chen, Parvez Ahammad, Colby Boyer, Shih-I Huang, Leon Lin, Edgar J. Lobaton, Marci Lenore Meingast, Songhwai Oh, Simon Wang, Posu Yan, Allen Yang, Chuohao Yeo, Lung-Chung Chang, J. D. Tygar, and S. Shankar Sastry, "CITRIC: A low-bandwidth wireless camera network platform," in *Third ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, September 2008.

[18] Mung Chiang, S.H. Low, A.R. Calderbank, and J.C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 255–312, Jan. 2007.

[19] Chipcon Products from Texas Instruments, *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, http://www.ti.com/lit/gpn/cc2420, March 2007, datasheet, Revision B.

[20] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. Cambridge, Massachusetts: MIT Press, 1990.

[21] Crossbow Technology, Incorporated, *TelosB Mote Platform*, http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf, December 2008, datasheet.

[22] David Culler, Deborah Estrin, and Mani Srivastava, "Overview of sensor networks," in *IEEE Computer, Special Issue on Sensor Networks*, August 2004.

[23] Swades De, Chunming Qiao, and Hongyi Wu, "Meshed multipath routing with selective forwarding: an efficient strategy in wireless sensor networks," *Computer Networks*, vol. 43, no. 4, pp. 481–497, 2003.

[24] L. Doherty, W. Lindsay, and J. Simon, "Channel-specific wireless sensor network path data," in *Proceedings of 16th International Conference on Computer Communications and Networks (ICCCN)*, Aug. 2007, pp. 89–94.

[25] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & Chemical Engineering*, vol. 17, pp. 245–255, Mar 1993.

[26] Richard Draves, Jitendra Padhye, and Brian Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *MobiCom '04: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*. New York, NY, USA: ACM, 2004, pp. 114–128.

[27] Henri Dubois-Ferriere, "Anypath routing," Ph.D. dissertation, EPFL, Lausanne, 2006. [Online]. Available: http://library.epfl.ch/theses/?nr=3636

[28] S. Dulman, T. Nieberg, J. Wu, and P. Havinga, "Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks," in *Proceedings of the Wireless Communications and Networking Conference*, 2003.

[29] Dust Networks, Inc., *SmartMesh-XT M2030 Product Specification*, http://www.dustnetworks.com/docs/M2030.pdf, 2006, datasheet.

[30] Energizer Holdings, Incorporated, *Energizer EN91 Product Datasheet*, http://data.energizer.com/PDFs/EN91.pdf, December 2008, AA Battery Datasheet.

[31] Sinem Coleri Ergen, "Wireless sensor networks: Energy efficiency, delay guarantee and fault tolerance," Ph.D. dissertation, University of California, Berkeley, 2005.

[32] Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukhatme, "Connecting the physical world with pervasive networks," in *IEEE Pervasive Computing*, vol. 1, 2002, pp. 59–69.

[33] K. D. Frampton, "Distributed group-based vibration control with a networked embedded system," *Smart Materials and Structures*, vol. 14, no. 2, pp. 307–314, April 2005.

[34] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," *SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 4, pp. 11–25, 2001.

[35] Yashar Ganjali and Abtin Keshavarzian, "Load balancing in ad hoc networks: single-path routing vs. multi-path routing," in *23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 2, 2004, pp. 1120–1125.

[36] D. J. Gaushell and H. T. Darlington, "Supervisory control and data acquisition," *Proceedings of the IEEE*, vol. 75, pp. 1645–1658, 1987.

[37] Chris Godsil and Gordon Royle, *Algebraic Graph Theory*. Springer, April 2001.

[38] W.K. Hale, "Frequency assignment: Theory and applications," *Proceedings of the IEEE*, vol. 68, no. 12, pp. 1497–1514, Dec. 1980.

[39] Jane K. Hart and Kirk Martinez, "Environmental sensor networks: A revolution in the earth system science?" *Earth-Science Reviews*, vol. 78, pp. 177–191, Oct 2006. [Online]. Available: http://www.sciencedirect.com/science/article/B6V62-4KBVWX7-2/2/9131e5d84373382ba80cb07e1eaece48

[40] HART Communication Foundation, *WirelessHART Data Sheet*, http://www.hartcomm2.org/hart_protocol/wireless_hart/wirelesshart_datasheet.pdf, 2007, datasheet.

[41] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, vol. 2, 2000, p. 10.

[42] Ted Herman and Sébastien Tixeuil, "A distributed TDMA slot assignment algorithm for wireless sensor networks," in *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, Sotiris Nikoletseas and José D. P. Rolim, Eds. Berlin; New York: Springer, July 2004.

[43] J. P. Hespanha, P. Naghshtabrizi, and Yonggang Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, pp. 138–162, 2007.

[44] Roger A. Horn and Charles R. Johnson, *Matrix Analysis*. New York: Cambridge University Press, 1999.

[45] International Society of Automation, "ISA-SP100 wireless systems for automation website," http://www.isa.org/isa100.

[46] International Water Association, "IWA task group on benchmarking of control strategies for waste water treatment plants (WWTPs)," http://www.benchmarkwwtp.org/, 2008.

[47] Y. Ji, H. J. Chizeck, X. Feng, and K. A. Loparo, "Stability and control of discrete-time jump linear systems," *Control Theory Advanced Technology*, vol. 7, no. 2, pp. 247–270, 1991.

[48] Chris Karlof, Yaping Li, and Joe Polastre, "ARRIVE: Algorithm for robust routing in volatile environments," University of California at Berkeley, Tech. Rep. UCB/CSD-03-1233, May 2002.

[49] Richard M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York: Plenum, 1972, pp. 85–103.

[50] Yan Ke and R. Sukthankar, "PCA-SIFT: a more distinctive representation for local image descriptors," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. II–506–II–513 Vol.2, June–July 2004.

[51] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*. ACM Press, April 2007, pp. 254–263.

[52] C.E. Koksal and H. Balakrishnan, "Quality-aware routing metrics for time-varying wireless mesh networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 11, pp. 1984–1994, Nov. 2006.

[53] Lakshman Krishnamurthy, Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Nandakishore Kushalnagar, Lama Nachman, and Mark Yarvis, "Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea," in *SenSys '05: Proceedings of the 3rd International Conference*

on Embedded Networked Sensor Systems. New York, NY, USA: ACM Press, 2005, pp. 64–75. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098926

[54] Fabian Kuhn and Roger Wattenhofer, "On the complexity of distributed graph coloring," in *PODC '06: Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM, 2006, pp. 7–15.

[55] *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, LAN/MAN Standards Committee of the IEEE Computer Society, 3 Park Avenue, New York, NY 10016-5997, USA, September 2006, 802.15.4 Standard.

[56] Ákos Lédeczi, András Nádas, Péter Völgyesi, György Balogh, Branislav Kusy, János Sallai, Gábor Pap, Sebestyén Dóra, Károly Molnár, Miklós Maróti, and Gyula Simon, "Countersniper system for urban warfare," *ACM Transactions on Sensor Networks*, vol. 1, no. 2, pp. 153–177, 2005.

[57] Kyung Chang Lee and Hong-Hee Lee, "Network-based fire-detection system via controller area network for smart home automation," *IEEE Transactions on Consumer Electronics*, vol. 50, pp. 1093–1100, 2004.

[58] S.-J. Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," in *Proceedings of the IEEE International Conference on Communications ICC*, vol. 10, 2001, pp. 3201–3205 vol.10.

[59] Philip Levis, Neil Patel, David Culler, and Scott Shenker, "Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *NSDI'04: Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 2–2.

[60] Jing Li and G. Y. Lazarou, "A bit-map-assisted energy-efficient MAC scheme for wireless sensor networks," in *Third International Symposium on Information Processing in Sensor Networks (IPSN)*, 2004, pp. 55–60.

[61] Qun Li and Daniela Rus, "Navigation protocols in sensor networks," *ACM Transactions on Sensor Networks*, vol. 1, no. 1, pp. 3–35, 2005.

[62] Feng-Li Lian, J. Moyne, and D. Tilbury, "Network design consideration for distributed control systems," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 2, pp. 297–307, Mar 2002.

[63] Xiaojun Lin and N. B. Shroff, "The impact of imperfect scheduling on cross-layer congestion control in wireless networks," *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 302–315, April 2006.

[64] Nathan Linial, "Locality in distributed graph algorithms," *SIAM Journal on Computing*, vol. 21, no. 1, pp. 193–201, 1992. [Online]. Available: http://link.aip.org/link/?SMJ/21/193/1

[65] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov 2004,

10.1023/B:VISI.0000029664.99615.94. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[66] Michael Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1036–1053, 1986. [Online]. Available: http://link.aip.org/link/?SMJ/15/1036/1

[67] Mateusz Malinowski, Matthew Moskwa, Mark Feldmeier, Mathew Laibowitz, and Joseph A. Paradiso, "CargoNet: a low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events," in *SenSys '07: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM, 2007, pp. 145–159.

[68] M. V. Marathe, H. Breu, H.B. Hunt, III, S. S. Ravi, and D. J. Rosenkrantz, "Simple heuristics for unit disk graphs," *Networks*, vol. 25, no. 2, pp. 59–68, March 1995.

[69] J. Marcuse, B. Menz, and J. R. Payne, "Servers in SCADA applications," *IEEE Transactions on Industry Applications*, vol. 33, pp. 1295–1299, 1997.

[70] M. Marina and S. Das, "On-demand multi path distance vector routing in ad hoc networks," *Ninth International Conference on Network Protocols (ICNP)*, p. 14, 2001.

[71] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM, 2004, pp. 39–49.

[72] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, pp. 403–419, Aug 2000, 10.1023/A:1008930403506. [Online]. Available: http://dx.doi.org/10.1023/A:1008930403506

[73] M. Mercangöz and F.J. Doyle, III, "Model-based control in the pulp and paper industry," *IEEE Control Systems Magazine*, vol. 26, no. 4, pp. 30–39, Aug. 2006.

[74] J. R. Moyne and D. M. Tilbury, "The emergence of industrial control networks for manufacturing control, diagnostics, and safety data," *Proceedings of the IEEE*, vol. 95, pp. 29–47, 2007.

[75] Asis Nasipuri, Robert Castañeda, and Samir R. Das, "Performance of multipath routing for on-demand protocols in mobile ad hoc networks," *Mobile Networks and Applications*, vol. 6, no. 4, pp. 339–349, 2001.

[76] Songhwai Oh, Luca Schenato, Phoebus Chen, and Shankar Sastry, "Tracking and coordination of multiple agents using sensor networks: System design, algorithms and experiments," *Proceedings of the IEEE*, vol. 95, pp. 234–254, 2007.

[77] C.H. Papadimitriou and I.C. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[78] P. G. Park, C. Fischione, A. Bonivento, K. H. Johansson, and A. Sangiovanni-Vincentelli, "Breath: A self-adapting protocol for wireless sensor networks in control

and automation," in *Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2008, pp. 323–331.

[79] D. Peleg, *Distributed computing : a locality-sensitive approach*. Philadelphia: Society for Industrial and Applied Mathematics, 2000.

[80] Kristofer S. J. Pister and Lance Doherty, "TSMP: Time synchronized mesh protocol," in *Proceedings of the IASTED International Symposium on Distributed Sensor Networks (DSN)*, November 2008.

[81] Joseph Polastre, Jason Hill, and David Culler, "Versatile low power media access for wireless sensor networks," in *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, November 2004, pp. 95–107.

[82] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks," in *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM Press, 2003, pp. 181–192.

[83] Subramanian Ramanathan, "Scheduling algorithms for multihop radio networks," Ph.D. dissertation, Dept. of Computer and Information Sciences, University of Delaware, 1992.

[84] Subramanian Ramanathan, "A unified framework and algorithm for channel assignment in wireless networks," *Wireless Networks*, vol. 5, pp. 81–94, Mar 1999, 10.1023/A:1019126406181. [Online]. Available: http://dx.doi.org/10.1023/A:1019126406181

[85] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min, "Z-MAC: a hybrid MAC for wireless sensor networks," in *SenSys '05: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM Press, 2005, pp. 90–101.

[86] N. Lawrence Ricker, "Tennessee eastman challenge archive," http://depts.washington.edu/control/LARRY/TE/download.html/, 2008.

[87] Jeffrey S. Rosenthal, "Convergence rates of Markov chains," *SIAM Review*, vol. 37, no. 3, pp. 387–405, 1995. [Online]. Available: http://citeseer.ist.psu.edu/article/rosenthal95convergence.html

[88] Sabyasachi Roy, Dimitrios Koutsonikolas, Saumitra Das, and Y. Charlie Hu, "High-throughput multicast routing metrics in wireless mesh networks," *Ad Hoc Networks*, vol. 6, pp. 878–899, Aug 2008. [Online]. Available: http://www.sciencedirect.com/science/article/B7576-4PB0PP4-2/2/3ec5c21efe6b52d3798a4a515e844f21

[89] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, and S. S. Sastry, "Foundations of control and estimation over lossy networks," *Proceedings of the IEEE*, vol. 95, pp. 163–187, 2007.

[90] Luca Schenato, "Optimal estimation in networked control systems subject to random delay and packet loss," in *Proceedings of the 45th IEEE Conference on Decision and Control (CDC)*, December 2006.

[91] M. Schneider, J. Evans, P. Wright, and D. Ziegler, "Designing a thermoelectrically powered wireless sensor network for monitoring aluminium smelters," *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering*, vol. 220, pp. 181–190, Oct 2006, 10.1243/09544089JPME67. [Online]. Available: http://dx.doi.org/10.1243/09544089JPME67

[92] P. Seiler and Raja Sengupta, "An H_∞ approach to networked control," *IEEE Transactions on Automatic Control*, vol. 50, pp. 356–364, 2005.

[93] A. Sen and M.L. Huson, "A new model for scheduling packet radio networks," *INFOCOM Proceedings of the 15th Conference on Computer Communications*, vol. 3, pp. 1116–1124, Mar 1996.

[94] Sergio D. Servetto and Guillermo Barrenetxea, "Constrained random walks on random graphs: routing algorithms for large scale wireless sensor networks," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002, pp. 12–21.

[95] Sangati Seth, Jerome P. Lynch, and Dawn M. Tilbury, "Wirelessly networked distributed controllers for real-time control of civil structures," in *Proceedings of the American Control Conference*, vol. 4. Piscataway, NJ 08855-1331, United States: Institute of Electrical and Electronics Engineers, June 8-10 2005, pp. 2946–2952.

[96] Daniel Sexton, Michael Mahony, Michael Lapinski, and Jay Werb, "Radio channel quality in industrial wireless sensor networks," in *Proceedings of the ISA/IEEE Sensors for Industry Conference (SIcon)*, February 2005.

[97] Gaurav Sharma, Ravi R. Mazumdar, and Ness B. Shroff, "On the complexity of scheduling in wireless networks," in *MobiCom '06: Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*. New York, NY, USA: ACM, 2006, pp. 227–238.

[98] Cory Sharp, Shawn Schaffert, Alec Woo, Naveen Sastry, Chris Karlof, Shankar Sastry, and David Culler, "Design and implementation of a sensor network system for vehicle tracking and autonomous interception," in *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN)*, January 2005, pp. 93–107.

[99] Victor Shnayder, Bor rong Chen, Konrad Lorincz, Thaddeus R. F. Fulford-Jones, and Matt Welsh, "Sensor networks for medical care," Division of Engineering and Applied Sciences, Harvard University, Tech. Rep. TR-08-05, Apr 2005. [Online]. Available: http://www.eecs.harvard.edu/~mdw/papers/codeblue-techrept05.pdf

[100] D. Snoonian, "Smart buildings," *IEEE Spectrum*, vol. 40, pp. 18–23, 2003.

[101] D. Steingart, J. Wilson, A. Redfern, P.K. Wright, R. Romero, and L. Lim, "Augmented cognition for fire emergency response: An iterative user study," in *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI)*, July 2005.

[102] William J. Stewart, *Introduction to the Numerical Solutions of Markov Chains*. Princeton, New Jersey: Princeton University Press, 1994.

[103] Jos F. Sturm, "Sedumi website," http://sedumi.mcmaster.ca/.

[104] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler, "An analysis of a large scale habitat monitoring application," in *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM Press, 2004, pp. 214–226. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031521

[105] Texas Instruments Incorporated, *MSP430x15x, MSP430x16x, MSP430x161x Mixed Signal Microcontroller*, http://www.ti.com/lit/gpn/msp430f1611, August 2006, datasheet.

[106] The MathWorks, Inc., "Model predictive control toolbox 3.0 - MPC supervisory control of a two stage thermo-mechanical pulping process demo," http://www.mathworks.com/products/mpc/demos.html?file=/products/demos/shipping/mpc/mpctmpdemo.html, 2008.

[107] Gilman Tolle, "A network management system for wireless sensor networks," Master's thesis, Univ. of California, Berkeley, 2005.

[108] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong, "A macroscope in the redwoods," in *SenSys '05: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM Press, 2005, pp. 51–63. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098925

[109] University of California, Berkeley and TinyOS Open Source Community, "TinyOS community forum," http://www.tinyos.net/.

[110] L. F. W. van Hoesel and P. J. M. Havinga, "A lightweight medium access protocol (LMAC) for wireless sensor networks," in *Proceedings of the International Workshop on Networked Sensing Systems (INSS)*, June 2004.

[111] W. Weber, J. Rabaey, and E. Aarts, Eds., *Ambient Intelligence*. Springer-Verlag, 2005, ch. How Ambient Intelligence will Improve Habitability and Energy Efficiency in Buildings, pp. 63–80, 10.1007/3-540-27139-2_5. [Online]. Available: http://dx.doi.org/10.1007/3-540-27139-2_5

[112] Joel Wilson, Dan Steingart, Russell Romero, Jessica Reynolds, Eric Mellers, Andrew Redfern, Lloyd Lim, William Watts, Colin Patton, Jessica Baker, and Paul Wright, "Design of monocular head-mounted displays for increased indoor firefighting safety and efficiency," *Helmet- and Head-Mounted Displays X: Technologies and Applications*, vol. 5800, no. 1, pp. 103–114, 2005. [Online]. Available: http://link.aip.org/link/?PSI/5800/103/1

[113] Alec Woo, Terence Tong, and David Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM, 2003, pp. 14–27.

[114] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin, "A wireless sensor network for structural monitoring," in *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM Press, 2004, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031498

[115] Yonggang Xu and J. P. Hespanha, "Estimation under uncontrolled and controlled communications in networked control systems," in *Proceedings of the 44th IEEE Conference on Decision and Control (CDC)*, 2005, pp. 842–847.

[116] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang, "GRAdient Broadcast: a robust data delivery protocol for large scale sensor networks," *Wireless Networks*, vol. 11, no. 3, pp. 285–298, 2005.

[117] Wei Ye, Fabio Silva, and John Heidemann, "Ultra-low duty cycle MAC with scheduled channel polling," in *SenSys '06: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM Press, 2006, pp. 321–334.

[118] Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi, "A framework for reliable routing in mobile ad hoc networks," in *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 1, 2003, pp. 270–280.

[119] ZigBee Alliance, Inc., *ZigBee Specification*, ZigBee Standards Organization, 2400 Camino Ramon, Suite 375; San Ramon, CA 94583, January 2007.

Appendix A

Math Notation Convention

This dissertation tries to follow the mathematical notation described below in all the chapters. Due to the large number of symbols, some symbols are reused in different sections of the text, but their meanings should be clear from the context. Occasionally, we will abuse the notation and use one symbol to denote two similar concepts.

- *variables* of one dimension are in lowercase, e.g., a
- numerical *constants* are in capital letters, e.g., N
- *events* from a sample space are also in capital letters, e.g., A
- *matrices* are also in capital letters, e.g., A
- *vectors* are in bold or have an arrow on top, e.g., $\vec{a}$, $\boldsymbol{a}$
- *random variables* are also in bold, e.g., $\boldsymbol{\theta}$
- *sets* are in calligraphic letters, e.g., $\mathcal{A}$

We often use superscripts and subscripts to label our variables. To avoid confusion with exponentiation, we sometimes use parentheses around superscripts when we use them for labeling. Note that we have both single subscripts (e.g., p_l) and double subscripts (e.g., p_{ij}) for labels, and in the latter we do not separate the subscripts with commas. When a double subscript involves an expression, we use parentheses around the expression for a subscript to separate the two subscripts (e.g., $p_{(N+1)i}$).

(a, b)	ordered pair (can represent a link, position in a matrix)
	open interval between a and b
$(a, b, \ldots)$	ordered list, sequences
$[a, b]$	closed interval between a and b
$[a\ b\ \cdots]$	vector
$[a_{ij}]$	matrix with element a_{ij} at position (i, j)
$\{a, b, \ldots\}$	set
$2^{\mathcal{A}}$	power set of set $\mathcal{A}$ (set of all subsets of $\mathcal{A}$)
$\forall a$	for all a

$\exists a$	there exists a
: or s.t.	such that
$a \in \mathcal{A}$	a is an element in the set $\mathcal{A}$
$\mathcal{A} \subseteq \mathcal{B}$	$\mathcal{A}$ is a subset of $\mathcal{B}$
$\mathcal{A} \cup \mathcal{B}$	the union of sets $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A} \cap \mathcal{B}$	the intersection of sets $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A} \backslash \mathcal{B}$	set difference, set with elements in $\mathcal{A}$ not in $\mathcal{B}$
$\lvert\mathcal{A}\rvert$	the cardinality of set $\mathcal{A}$ (number of elements in $\mathcal{A}$)
$\mathbb{N}$	natural numbers including 0, namely $\{0, 1, 2, \ldots\}$
$\mathbb{N}_1$	positive integers, namely $\{1, 2, \ldots\}$
$\mathbb{Z}$	integers
$\mathbb{R}$	real numbers
$\mathbb{R}_+$	nonnegative real numbers
$\mathbb{C}$	complex numbers
$\mathbb{R}^n$	set of n-tuples (n-dimensional vectors) with elements in $\mathbb{R}$
$\mathbb{R}^{m \times n}$	set of $m \times n$ arrays with entries in $\mathbb{R}$
$f : \mathcal{A} \mapsto \mathcal{B}$	function f mapping domain $\mathcal{A}$ into codomain $\mathcal{B}$
A^T	transpose of matrix A
A^*	complex conjugate transpose of matrix A
$\lVert A\rVert$	norm of matrix A
$\lVert\boldsymbol{a}\rVert$	norm of vector $\boldsymbol{a}$
$\lVert\boldsymbol{a}\rVert_1$	ℓ_1-norm of a vector $\boldsymbol{a}$, $\sum_i \lvert\boldsymbol{a}_i\rvert$
$\mathrm{diag}(\boldsymbol{a})$	matrix consisting of the elements of $\boldsymbol{a}$ on the main diagonal and 0 everywhere else
$\mathbb{E}[\cdot]$	expectation
$\mathbb{P}(\cdot)$	probability
$\mathbb{I}(\cdot)$	indicator function
$O(\cdot)$	big-O notation for computational complexity
$EO(\cdot)$	expected computational complexity is $O(\cdot)$
$x \triangleq y$	x is defined to be y
$x := y$	x is assigned the value of y

The notation above may sometimes be combined to form expressions that look nonconventional. For instance, $[0, 1]^{N \times N}$ is a set of $N \times N$ arrays with elements that lie within the interval $[0, 1]$, while $\{0, 1\}^N$ is a set of N-dimensional vectors with elements that are either 0 or 1.

Appendix B

Full DSFMC Model

The following is the DSFMC model without the assumption that links within a pair of stages $(k, k+1)$ are independent.

Definition B.0.1 (Directed Staged Flooding Markov Chain Model)**.** Let's assume we have a routing topology with $K+1$ stages $0, \ldots, K$. Each stage k has N_k nodes, and the set of 2^{N_k} possible states in stage k is represented by the set of numbers $\mathcal{S}^{(k)} = \{0, \ldots, 2^{N_k} - 1\}$. Let $\mathcal{K}^{(k)}$ be the set of nodes in stage k and for each state $\sigma^{(k)} \in \mathcal{S}^{(k)}$, let $\mathcal{R}_\sigma^{(k)} \subset \mathcal{K}^{(k)}$ be the set of nodes that have received a copy of the packet and $\mathcal{U}_\sigma^{(k)} = \mathcal{K}^{(k)} \backslash \mathcal{R}_\sigma^{(k)}$ be the set of nodes that have not received a copy of the packet (See Figure 4.10). Let $\omega^{(k)}$ denote the state where no nodes received a copy of the packet in stage k.

Let $R_\sigma^{(k)}$ denote the event that only the nodes in $\mathcal{R}_\sigma^{(k)}$ received a copy of the packet, $S_{(i,j)}$ denote the event a packet was at node i and link (i,j) successfully transmitted the packet, and $\bar{S}_{(i,j)}$ denote the event that a packet was at node i but link (i,j) failed.[1] The conditional probability of the next state $\boldsymbol{X}^{(k+1)}$ being in state $\sigma^{(k+1)}$ given that the current state $\boldsymbol{X}^{(k)}$ is $\sigma^{(k)}$ can be expressed in terms of these events as

$$\mathbb{P}(\boldsymbol{X}^{(k+1)} = \sigma^{(k+1)} | \boldsymbol{X}^{(k)} = \omega^{(k)}) = \begin{cases} 1 & : \quad \sigma^{(k+1)} = \omega^{(k+1)} \\ 0 & : \quad \text{otherwise} \end{cases}$$

if $\sigma^{(k)} \neq \omega^{(k)}$

$$\mathbb{P}(\boldsymbol{X}^{(k+1)} = \sigma^{(k+1)} | \boldsymbol{X}^{(k)} = \sigma^{(k)}) = \mathbb{P}\left(\bigcap_{u^{(k+1)} \in \mathcal{U}_\sigma^{(k+1)}} \left(\bigcap_{r^{(k)} \in \mathcal{R}_\sigma^{(k)}} \bar{S}_{(r^{(k)}, u^{(k+1)})} \right) \cap \bigcap_{r^{(k+1)} \in \mathcal{R}_\sigma^{(k+1)}} \overline{\left(\bigcap_{r^{(k)} \in \mathcal{R}_\sigma^{(k)}} \bar{S}_{(r^{(k)}, r^{(k+1)})} \right)} \middle| R_\sigma^{(k)} \right) \tag{B.1}$$

where the overbar denotes taking the complement of an event. The transition probability matrices between stage k and $k+1$ are $P^{(k+1)} \in [0,1]^{N_{k+1} \times N_k}$, where the entry in position $(\sigma^{(k+1)}, \sigma^{(k)})$ of the matrix is $\mathbb{P}(\boldsymbol{X}^{(k+1)} = \sigma^{(k+1)} | \boldsymbol{X}^{(k)} = \sigma^{(k)})$.

[1] The event $S_{(i,j)}$ is empty (and occurs with probability 0) if link (i,j) does not exist.

The initial state $\boldsymbol{X}^{(0)}$ is the state $\sigma^{(0)}$ corresponding to $\mathcal{R}_\sigma^{(0)} = \{a\}$, where a is the source node. Then, the probability distribution $\boldsymbol{p}^{(k)} \in [0,1]^{N_k}$ of the state at stage k is

$$\boldsymbol{p}^{(k)} = \underbrace{P^{(k)} \cdots P^{(2)} P^{(1)}}_{P^{(\underline{k})}} \boldsymbol{p}^{(0)} \tag{B.2}$$

□

Appendix C

Proofs of Theorems 4.2.1 and 4.3.1

The proofs of Theorems 4.2.1 and 4.3.1 rely heavily on the following theorem:

Theorem C.0.1 (ρ_* determines convergence rate of $\boldsymbol{p}^{(t)}$)**.** *Let $P \in [0,1]^{N\times N}$ be a column stochastic matrix (meaning all the entries in the matrix are nonnegative and all the columns sum to 1) with $\lim_{k\to\infty} P^k\boldsymbol{p} = \boldsymbol{e}^{[b]}$ for all probability vectors $\boldsymbol{p} \in [0,1]^N, \sum_i \boldsymbol{p}_i = 1$. Here, $\boldsymbol{e}^{[b]}$ is an elementary vector with the b-th element equal to 1 and all other elements equal to 0. Let $\rho_* = \max\{|\lambda| : \lambda$ is an eigenvalue of P and $|\lambda| < 1\}$. Then,*

$$\left\|P^k\boldsymbol{p} - \boldsymbol{e}^{[b]}\right\|_1 \leq Ck^{J-1}(\rho_*)^{k-J+1}, \quad \forall k \in \mathbb{N}_1 \tag{C.1}$$

where C is a constant dependent on $\boldsymbol{p}$, and $J \in \mathbb{N}_1$ is the size of the largest Jordan block of P. □

The proof of this theorem is given later in Appendix C.3.

C.1 Proof of Theorem 4.2.1

Proof.

$$\begin{aligned}
\left\|(P^{(\underline{T})})^k\boldsymbol{p}^{(0)} - \boldsymbol{e}^{[b]}\right\|_1 &= \left(\sum_{j\neq b}\left|\boldsymbol{p}_j^{(Tk)}\right|\right) + \left|\boldsymbol{p}_b^{(Tk)} - 1\right| \\
&\overset{(a)}{=} \sum_{j\neq b}\left|\boldsymbol{p}_j^{(Tk)}\right| + \left|-\sum_{j\neq b}\boldsymbol{p}_j^{(Tk)}\right| \\
&\overset{(b)}{=} 2\sum_{j\neq b}\boldsymbol{p}_j^{(Tk)}
\end{aligned}$$

where step (a) uses the relationship $\boldsymbol{p}_b^{(t_d)} = 1 - \sum_{j\neq b}\boldsymbol{p}_j^{(t_d)}$ and step (b) uses the fact that the $\boldsymbol{p}_j^{(Tk)}$ are nonnegative.

By applying Theorem C.0.1 to $P^{(\underline{T})}$ we see that

$$\left\|(P^{(\underline{T})})^k\boldsymbol{p}^{(0)} - \boldsymbol{e}^{[b]}\right\|_1 \leq Ck^{J-1}(\rho_*)^{k-J+1}, \quad \forall k \in \mathbb{N}_1 \quad .$$

If we let $k = \lfloor \frac{t_d}{T} \rfloor$, we can combine the steps above to get

$$\begin{aligned} p_{\text{net}}^{(t_d)} &= \boldsymbol{p}_b^{(t_d)} \overset{(c)}{\geq} \boldsymbol{p}_b^{(Tk)} = 1 - \sum_{j \neq b} \boldsymbol{p}_j^{(Tk)} \\ &= 1 - \frac{1}{2} \left\| (P^{(\underline{T})})^k \boldsymbol{p}^{(0)} - \boldsymbol{e}^{[b]} \right\|_1 \\ &\geq 1 - \frac{1}{2} C k^{J-1} (\rho_*)^{k-J+1} \end{aligned}$$

where step (c) comes from the fact that b is an absorbing state in the Markov chain.

To summarize,

$$p_{\text{net}}^{(t_d)} \geq 1 - C k^{J-1} (\rho_*)^{k-J+1}, \quad k = \left\lfloor \frac{t_d}{T} \right\rfloor \tag{C.2}$$

for some constant C dependent on the initial distribution $\boldsymbol{p}^{(0)}$ and $J \in \mathbb{N}_1$ the size of the largest Jordan block of $P^{(\underline{T})}$. Therefore, $p_{\text{net}}^{(t_d)}$ converges to 1 exponentially with a rate ρ_*. □

C.2 Proof of Theorem 4.3.1

The steps in this proof are similar to the steps in the proof of Theorem 4.2.1.

Proof.

$$\begin{aligned} \left\| P^K \boldsymbol{p}^{(0)} - \boldsymbol{e}^{[\omega]} \right\|_1 &= \left(\sum_{j \neq \omega} \left| \boldsymbol{p}_j^{(K)} \right| \right) + \left| \boldsymbol{p}_\omega^{(K)} - 1 \right| \\ &= \sum_{j \neq \omega} \left| \boldsymbol{p}_j^{(K)} \right| + \left| - \sum_{j \neq \omega} \boldsymbol{p}_j^{(K)} \right| \\ &= 2 \sum_{j \neq \omega} \boldsymbol{p}_j^{(K)} \end{aligned}$$

By applying Theorem C.0.1 to P we see that

$$\left\| P^K \boldsymbol{p}^{(0)} - \boldsymbol{e}^{[\omega]} \right\|_1 \leq C K^{J-1} (\rho_*)^{K-J+1}, \quad \forall K \in \mathbb{N}_1 \quad .$$

Letting $t_d = K N_{\text{stage}}$ and combining the steps above, we get

$$\begin{aligned} p_{\text{net}}^{(t_d)} &= 1 - \boldsymbol{p}_\omega^{(K)} = \sum_{j \neq \omega} \boldsymbol{p}_j^{(K)} \\ &= \frac{1}{2} \left\| P^K \boldsymbol{p}^{(0)} - \boldsymbol{e}^{[\omega]} \right\|_1 \\ &\leq \frac{1}{2} C K^{J-1} (\rho_*)^{K-J+1} \end{aligned}$$

To summarize,

$$p_{\text{net}}^{(t_d)} \leq C K^{J-1} (\rho_*)^{K-J+1}, \quad t_d = K N_{\text{stage}} \tag{C.3}$$

for some constant C dependent on the initial distribution $\boldsymbol{p}^{(0)}$ and $J \in \mathbb{N}_1$ the size of the largest Jordan block of P. Therefore, $p_{\text{net}}^{(t_d)}$ converges to 0 exponentially with a rate ρ_*. □

C.3 Proof of Theorem C.0.1

C.3.1 Statement of Theorems and Lemmas Used in Proof

First, we state some theorems and definitions used in the proof, with the notation modified from their original sources to stay consistent with the notation used throughout this dissertation. Theorems C.3.1 and C.3.2 are not used explicitly in the proof, but are stated for the reader to better grasp Theorem C.3.3.

Theorem C.3.1 (Theorem 5.6.9 from [44])**.** *If $\|\cdot\|$ is any matrix norm and if $A \in \mathbb{C}^{N\times N}$, then $\rho(A) \leq \|A\|$, where $\rho(A) \triangleq \max\{|\lambda| : \lambda \text{ is an eigenvalue of } A\}$ is the spectral radius of A.* □

Theorem C.3.2 (Spectral radius of a stochastic matrix)**.** *The spectral radius (magnitude of the maximum eigenvalue) of a column stochastic matrix P is 1.* □

Proof. A proof of this can be be found in [102], and is reproduced here.

Since P is a column stochastic matrix, $\sum_{i=1}^{N} P_{ij} = 1$ for all j. This means

$$\|P\|_1 \triangleq \max_j \sum_{i=1}^{N} |P_{ij}| = \max_j \sum_{i=1}^{N} P_{ij} = 1$$

where $\|\cdot\|_1$ is the *maximum column sum norm*, and the first equality holds because $P_{ij} \geq 0$ for all i and j. Combining this with Theorem C.3.1, we see that $\rho(P) \leq 1$. □

Definition C.3.1 (Periodic Markov chains, from [7])**.** A Markov chain is *periodic* if its states can be grouped into $K > 1$ disjoint subsets $S_1, \ldots, S_K$ so that

$$\begin{array}{l} \text{if } i \in S_k \text{ and } p_{ij} > 0, \\ \text{then } \begin{cases} j \in S_{k+1}, & \text{if } k = 1, \ldots, K-1 \\ j \in S_1, & \text{if } k = K \end{cases} \end{array}.$$

A Markov chain is *aperiodic* if it is not periodic. □

Definition C.3.2 (Decomposable Markov chains, from [87])**.** A Markov chain is *decomposable* if the state space $\mathcal{S}$ contains two non-empty disjoint subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ which are closed, i.e., such that the probability that $i \in \mathcal{S}_1$ transitions to another node in $\mathcal{S}_1$ is 1 and the probability that $j \in \mathcal{S}_2$ transitions to another node in $\mathcal{S}_2$ is 1. □

For the theorem below from Rosenthal, let $\lambda_0 = 1$ (the trivial eigenvalue of P) and $\rho_* = \max_{1\leq j\leq n-1} |\lambda_j|$, the largest absolute value of the nontrivial eigenvalues of P. From the theorem, we can also say that $\rho_* = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } P \text{ and } |\lambda| < 1\}$, which is used in the statement of the theorems of this dissertation. Other papers often refer to ρ_* as the second largest eigenvalue of the transition probability matrix.

Theorem C.3.3 (Fact 4 from [87]). *A finite Markov chain satisfies $\rho_* < 1$ if and only if it is both indecomposable and aperiodic.* □

For the theorem below from Rosenthal, let the *total variation distance* between probability measures v_1 and v_2 be defined as $\|v_1 - v_2\|_{\text{var}} \triangleq \sup_{A \subset \mathcal{S}} |v_1(A) - v_2(A)|$. Then, if $\mathcal{S}$ is finite, $\|v_1 - v_2\|_{\text{var}} = \frac{1}{2} \sum_{i \in \mathcal{S}} |v_1(i) - v_2(i)|$.

Theorem C.3.4 (Part of Fact 3 from [87]). *Suppose P satisfies $\rho_* < 1$ and the state space $\mathcal{S}$ is finite. Then, there is a unique stationary distribution $\boldsymbol{\pi}$ on $\mathcal{S}$ and, given an initial distribution $\boldsymbol{p}^{(0)}$ and point $i \in \mathcal{S}$, there is a constant $C_i > 0$ such that*

$$|\boldsymbol{p}_i^{(k)} - \boldsymbol{\pi}_i| \leq C_i k^{J-1} (\rho_*)^{k-J+1}$$

where J is the size of the largest Jordan block of P. It follows immediately that

$$\|\boldsymbol{p}^{(k)} - \boldsymbol{\pi}\|_{\text{var}} \leq C k^{J-1} (\rho_*)^{k-J+1} \tag{C.4}$$

where $C = \frac{1}{2} \sum C_i$. In particular, if P is diagonalizable (so that $J = 1$) then

$$\begin{aligned} \|\boldsymbol{p}_i^{(k)} - \boldsymbol{\pi}_i\|_{\text{var}} &\leq \sum_{m=1}^{n-1} |a_m \boldsymbol{v}_m(i)| |\lambda_m|^k \\ &\leq \left(\sum_{m=1}^{n-1} |a_m \boldsymbol{v}_m(i)| \right) (\rho_*)^k \end{aligned}$$

where $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{n-1}$ are a basis of right eigenvectors corresponding to $\lambda_0, \ldots, \lambda_{n-1}$ respectively, and where a_m are the (unique) complex coefficients satisfying

$$\boldsymbol{p}^{(0)} = a_0 \boldsymbol{v}_0 + a_1 \boldsymbol{v}_1 + \cdots + a_{n-1} \boldsymbol{v}_{n-1} \quad .$$

Here, $\boldsymbol{v}_m(i)$ denotes the i-th coordinate of the vector $\boldsymbol{v}_m$. □

For finite $\mathcal{S}$, we can relate the 1-norm to the total variation distance by

$$\|v_1 - v_2\|_{\text{var}} = \frac{1}{2} \sum_i |v_1(i) - v_2(i)| = \frac{1}{2} \|v_1 - v_2\|_1 \quad . \tag{C.5}$$

This means that (C.4) can be restated as

$$\|\boldsymbol{p}^{(k)} - \boldsymbol{\pi}\|_1 \leq C k^{J-1} (\rho_*)^{k-J+1} \tag{C.6}$$

where $C = \sum C_i$.

C.3.2 Proof of Theorem C.0.1

Proof. A column stochastic matrix $P \in [0,1]^{N \times N}$ with $\lim_{k \to \infty} P^k \boldsymbol{p} = \boldsymbol{e}^{[b]}$ for all probability vectors $\boldsymbol{p} \in [0,1]^N, \sum_i \boldsymbol{p}_i = 1$, describes the transition probability matrix for a Markov chain that is both indecomposable and aperiodic. The Markov chain is not decomposable because a decomposable Markov chain has more than one stationary distribution, whereas the Markov chain described by P has a unique stationary distribution $\boldsymbol{e}^{[b]}$. For instance, a

decomposable Markov chain would have a stationary probability distribution with nonzero entries over only the states in $\mathcal{S}_1$, and another stationary probability distribution with nonzero entries over only the states in $\mathcal{S}_2$. The Markov chain described by P is aperiodic because all probability distributions converge to a unique stationary distribution, meaning that there is no distribution that transitions in a periodic manner over time.

Since the Markov chain described by P is both indecomposable and aperiodic, we can apply Theorem C.3.3 and Theorem C.3.4 to get the desired result, where $\boldsymbol{p}^{(k)}$ corresponds to $P^k\boldsymbol{p}$ and $\boldsymbol{\pi}$ corresponds to $\boldsymbol{e}^{[b]}$. □

C.3.3 Discussion

The proof of Theorem C.0.1 appears to rely heavily on the assumption $\lim_{k\to\infty} P^k\boldsymbol{p} = \boldsymbol{e}^{[b]}$ for all probability vectors $\boldsymbol{p} \in [0,1]^N, \sum_i \boldsymbol{p}_i = 1$. For the UPDMC model, this corresponds to modeling a routing topology with a unique sink node where all packets are eventually routed to this sink. If we wish to apply this theorem to mesh networks with multiple collection points (sink nodes), as mentioned in Section 4.2.3, we need to make some simple modifications to the Markov chain model.

First, we would combine the states $i \in \mathcal{B}$ representing the sink nodes into one state $i_{\mathcal{B}}$ in the UPDMC model. The transition probabilities to this new state $i_{\mathcal{B}}$ would be $p_{ii_{\mathcal{B}}} = \sum_{j\in\mathcal{B}} p_{ij}$ while the transition probabilities $p_{i_{\mathcal{B}}j}$ out of $i_{\mathcal{B}}$ would be

$$p_{i_{\mathcal{B}}j} = \begin{cases} 1 & : \quad j = i_{\mathcal{B}} \\ 0 & : \quad j \neq i_{\mathcal{B}} \end{cases}$$

meaning $i_{\mathcal{B}}$ is a recurrent state. We can now apply Theorem C.0.1 to this new Markov chain model to show that the model converges to $i_{\mathcal{B}}$ at rate ρ_*. This means that the packet will eventually reach one of the sink nodes at rate ρ_*, although the packet arrival probability distribution over the nodes in $\mathcal{B}$ may depend on which node originally sent the packet.

Appendix D

Full-sized Transition Matrix for Figure 4.8

$$\begin{bmatrix}
1 & \bar{p}^2 & \bar{p}^3 & \bar{p}^5 & \bar{p}^2 & \bar{p}^4 & \bar{p}^5 & \bar{p}^7 \\
0 & p\bar{p} & p\bar{p}^2 & (1-\bar{p}^2)\bar{p}^3 & 0 & p\bar{p}^3 & p\bar{p}^4 & (1-\bar{p}^2)\bar{p}^5 \\
0 & p\bar{p} & p\bar{p}^2 & (1-\bar{p}^2)\bar{p}^3 & p\bar{p} & (1-\bar{p}^2)\bar{p}^2 & (1-\bar{p}^2)\bar{p}^3 & (1-\bar{p}^3)\bar{p}^4 \\
0 & p^2 & p^2\bar{p} & (1-\bar{p}^2)^2\bar{p} & 0 & p(1-\bar{p}^2)\bar{p} & p(1-\bar{p}^2)\bar{p}^2 & (1-\bar{p}^2)(1-\bar{p}^3)\bar{p}^2 \\
0 & 0 & p\bar{p}^2 & p\bar{p}^4 & p\bar{p} & p\bar{p}^3 & (1-\bar{p}^2)\bar{p}^3 & (1-\bar{p}^2)\bar{p}^5 \\
0 & 0 & p^2\bar{p} & p(1-\bar{p}^2)\bar{p}^2 & 0 & p^2\bar{p}^2 & p(1-\bar{p}^2)\bar{p}^2 & (1-\bar{p}^2)^2\bar{p}^3 \\
0 & 0 & p^2\bar{p} & p(1-\bar{p}^2)\bar{p}^2 & p^2 & p(1-\bar{p}^2)\bar{p} & (1-\bar{p}^2)^2\bar{p} & (1-\bar{p}^2)(1-\bar{p}^3)\bar{p}^2 \\
0 & 0 & p^3 & p(1-\bar{p}^2)^2 & 0 & p^2(1-\bar{p}^2) & p(1-\bar{p}^2)^2 & (1-\bar{p}^2)^2(1-\bar{p}^3)
\end{bmatrix}$$

Appendix E

Case Study Schedules and Models

E.1 UPD schedule for Camera Network

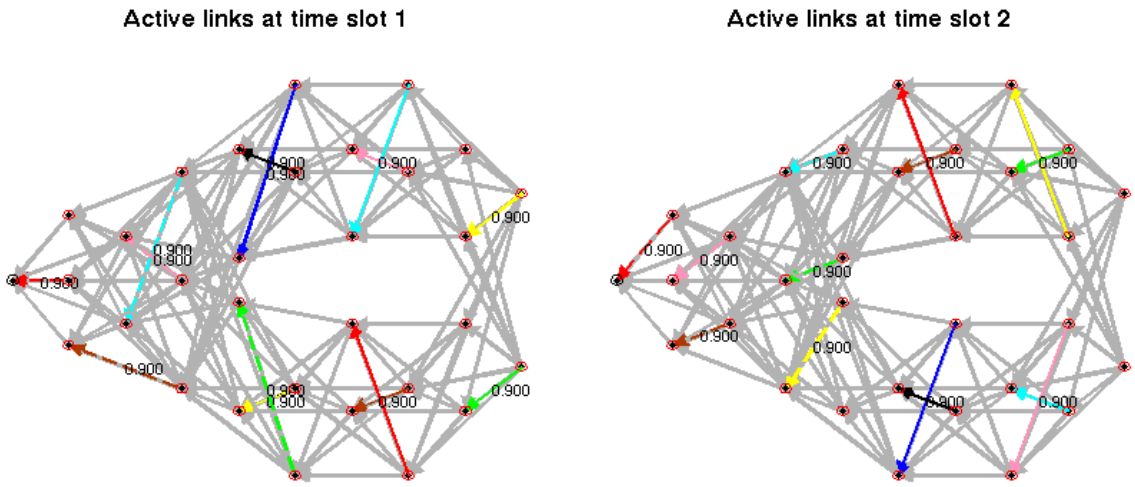

Figure E.1. UPD schedule time slots 1–2 for the Building Surveillance Case Study in Section 4.5.1. Nodes are laid out in the same position as the nodes in Figure 4.19 (node labels are omitted for clarity). Light gray arrows indicate links in the routing topology and all other arrows indicate links scheduled in the time slot. Links are labeled with their link probability.

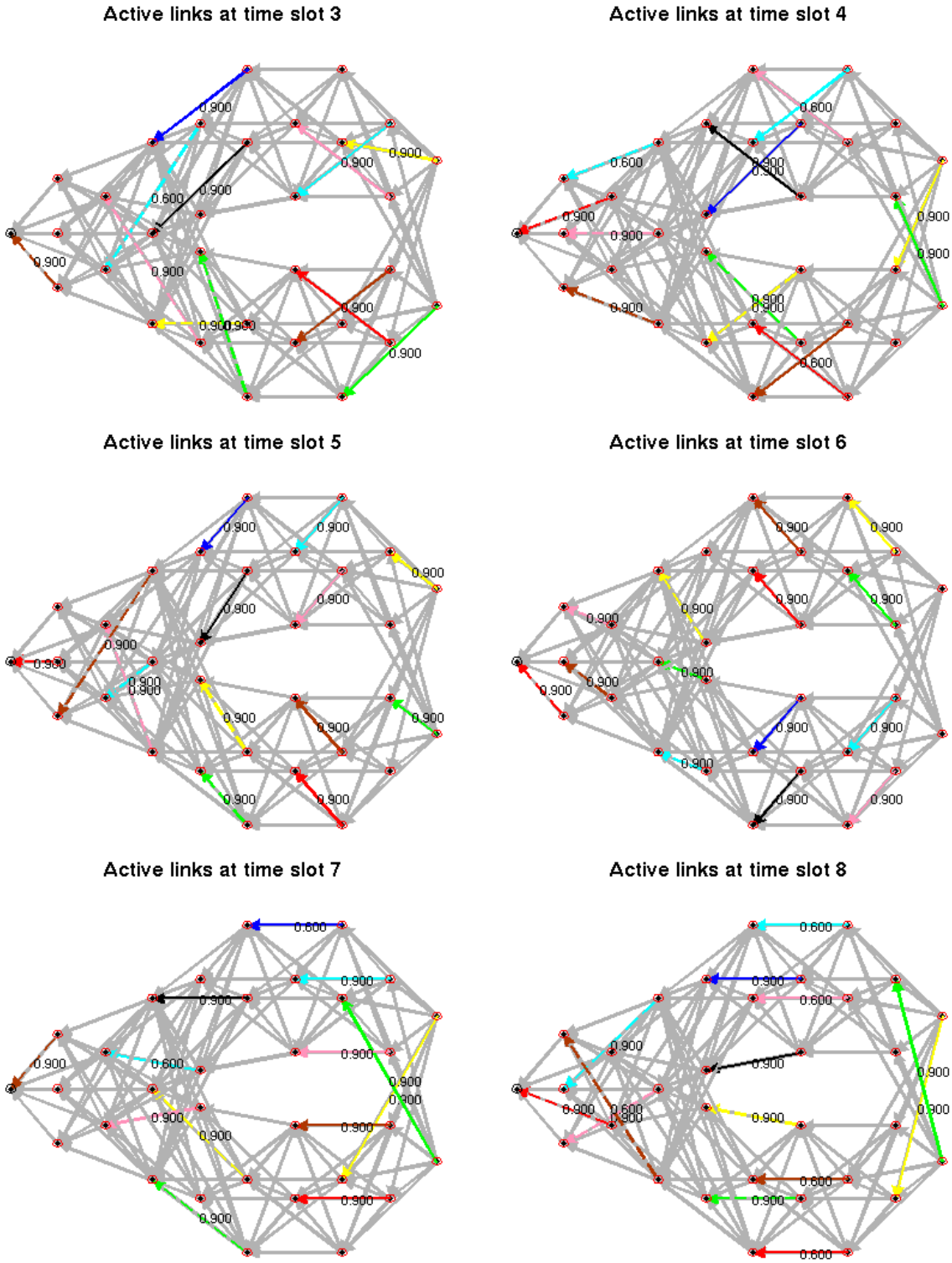

Figure E.2. UPD schedule time slots 3–8 for the Building Surveillance Case Study in Section 4.5.1. Nodes are laid out in the same position as the nodes in Figure 4.19 (node labels are omitted for clarity). Light gray arrows indicate links in the routing topology and all other arrows indicate links scheduled in the time slot. Links are labeled with their link probability.

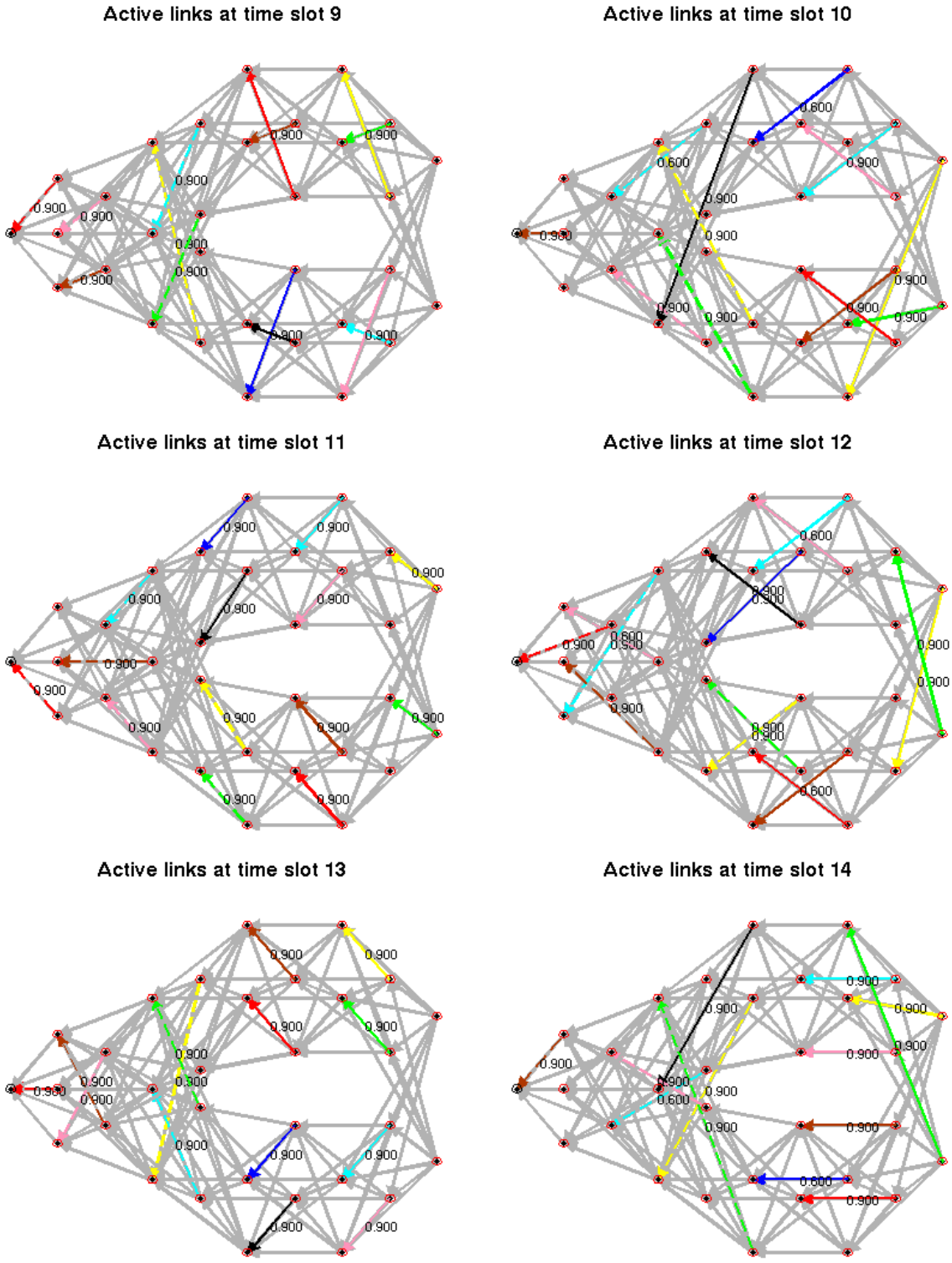

Figure E.3. UPD schedule time slots 9–14 for the Building Surveillance Case Study in Section 4.5.1. Nodes are laid out in the same position as the nodes in Figure 4.19 (node labels are omitted for clarity). Light gray arrows indicate links in the routing topology and all other arrows indicate links scheduled in the time slot. Links are labeled with their link probability.

E.2 DSF Schedule for Camera Network

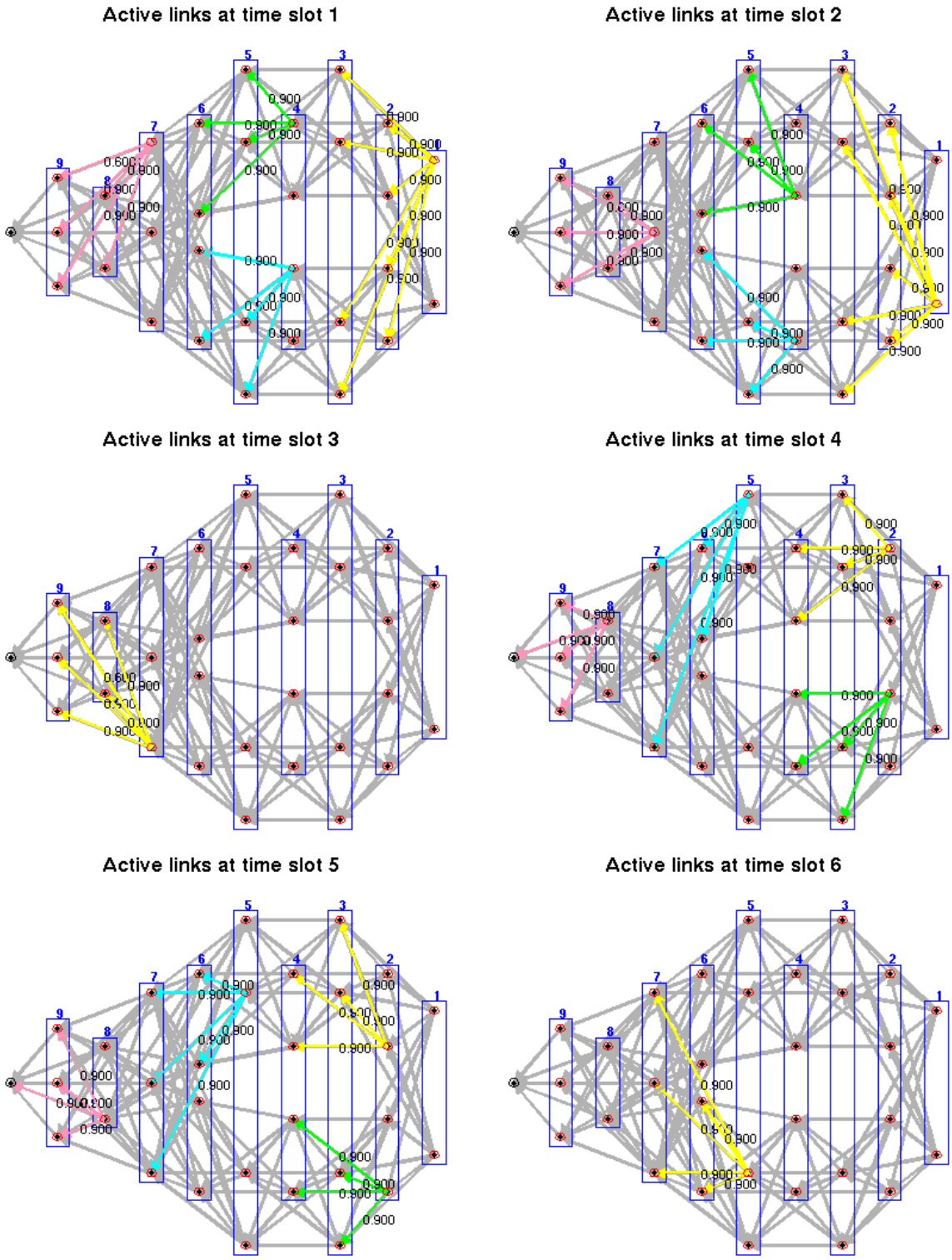

Figure E.4. DSF schedule time slots 1–6 for the Building Surveillance Case Study in Section 4.5.1. Nodes are laid out in the same position as the nodes in Figure 4.19 (node labels are omitted for clarity). Light gray arrows indicate links in the routing topology and all other arrows indicate links scheduled in the time slot. Links are labeled with their link probability.

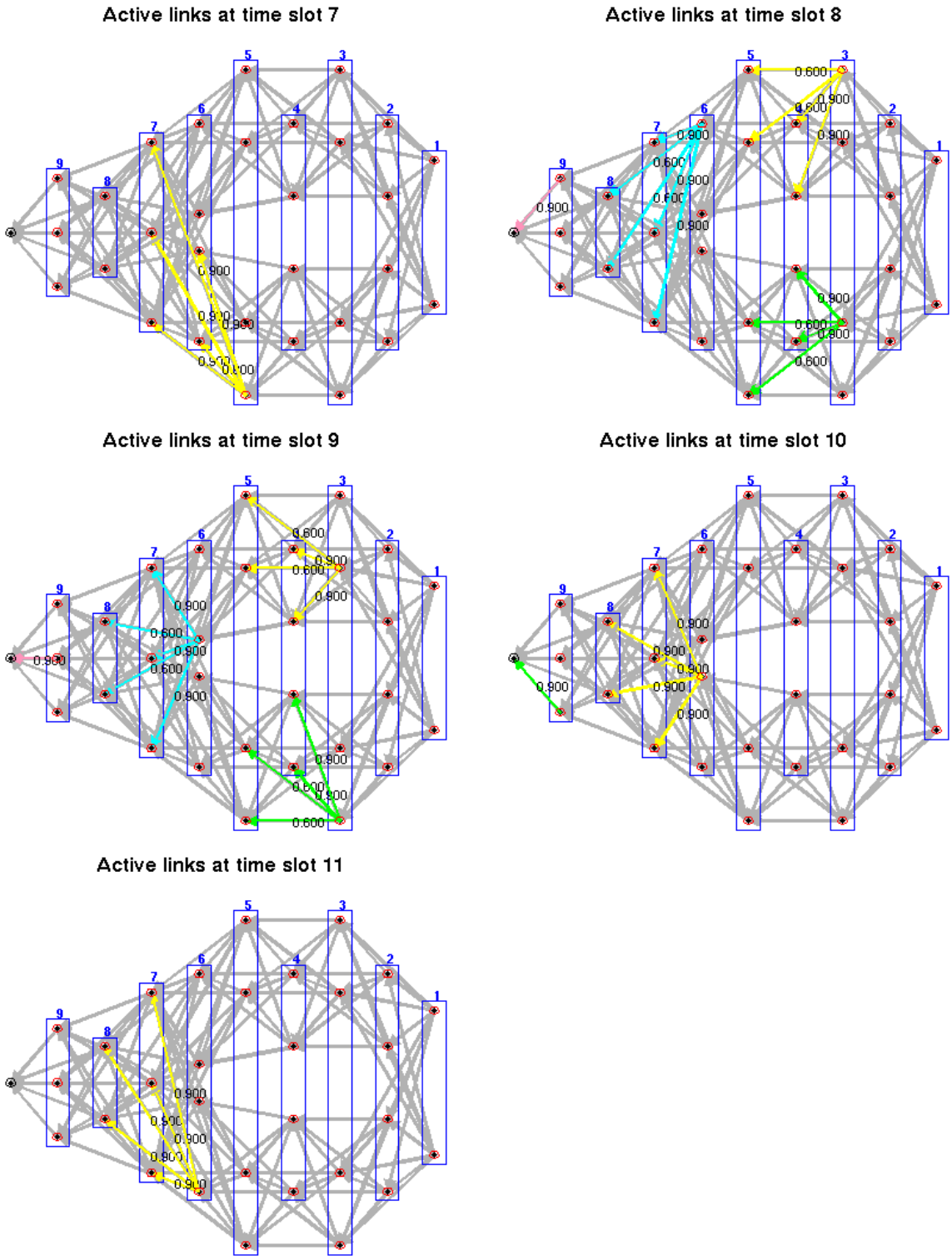

Figure E.5. DSF schedule time slots 7–11 for the Building Surveillance Case Study in Section 4.5.1. Nodes are laid out in the same position as the nodes in Figure 4.19 (node labels are omitted for clarity). Light gray arrows indicate links in the routing topology and all other arrows indicate links scheduled in the time slot. Links are labeled with their link probability.

E.3 Simulink Model for Pulp Mill NCS

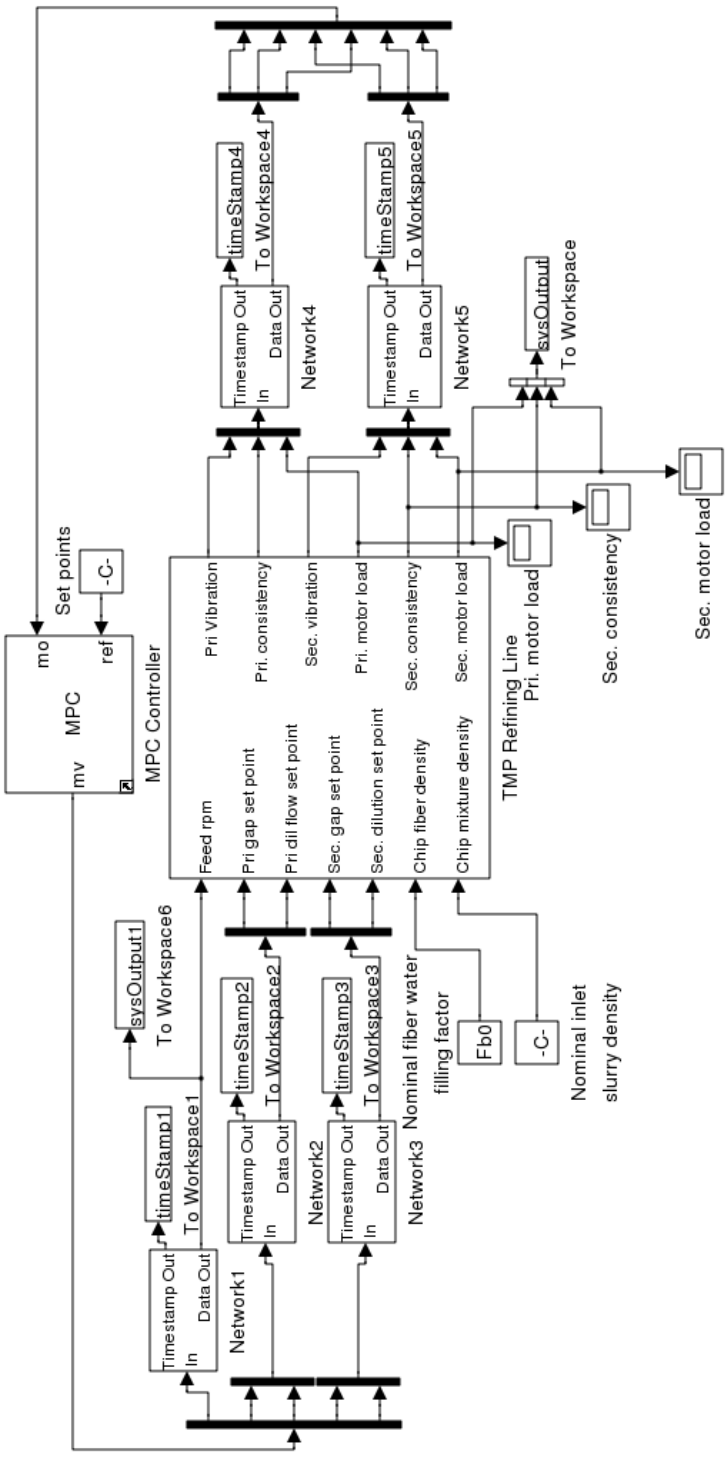

Figure E.6. Simulink diagram of the Pulp Mill Plant and MPC controller described in Section 4.5.2 connected with five lossy, delayed channels (one for each session). The packet drops and delays used by the *Network* blocks come from the MATLAB workspace. They come from a simulation of the wireless network using the topologies shown in Figure 4.33 and schedule shown in Figure 4.34.